VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

KONFERENČNÍ INFORMAČNÍ SYSTÉM

CONFERENCE INFORMATION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN BEDNÁŘ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/21013/2017/xbedna60

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Bednář Martin**
Obor: Informační technologie
Téma: **Konferenční informační systém**
Conference Information System
Kategorie: Web

Pokyny:

1. Seznamte se s platformou .NET pro tvorbu webových aplikací včetně webových rámců nad platformou .NET (např. ASP.NET, dotVVM, ad.).
2. Analyzujte požadavky na konferenční systém pro neziskovou organizaci Zámecké návrší, p. o., včetně nefunkčních požadavků, jako responzivita aplikace apod.
3. Proveďte výběr z nastudovaných webových rámců s ohledem na předchozí analýzu požadavků a navrhněte požadovanou aplikaci.
4. Dle návrhu aplikaci implementujte. Při implementaci se pokuste využít testy řízený vývoj i pro grafické uživatelské rozhraní (např. využitím automatického testování uživatelského rozhraní).
5. Výslednou aplikaci nasaďte, otestuje v reálném provozu a učiňte návrhy na další vývoj aplikace.

Literatura:

- *DotVVM - Open source MVVM framework for Web Apps* [online]. Praha: RIGANTI, s.r.o., 2017 [cit. 2017-10-23]. Dostupné z: https://www.dotvvm.com/
- Alex Homer: Professional ASP.NET web forms techniques. Wrox Press, Birmingham, 2002.
- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Křivka Zbyněk, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Platforma .NET nabízí množství rámců, technologií a přístupů pro vývoj webových aplikací. Orientace v nich je základem pro výběr efektivních implementačních technologií. Cílem práce bylo vytvořit konferenční informační systém jako webovou aplikaci na platformě .NET pro neziskovou organizaci Zámecké návrší, p.o. včetně zajištění jejího nasazení a provozu v produkčním prostředí. Před zahájením vývoje bylo nutné seznámit se s platformou .NET a porovnat rámce pro webový vývoj. V práci je představena vhodnost jednotlivých rámců pro různé druhy aplikací. Na základě analýzy prostředí .NET byl pro implementaci konferenčního informačního systému vybrán rámec DotVVM. Vývoj aplikační logiky probíhal metodou testy řízený vývoj. Díky kombinaci vhodných rámců, technologií a přístupů bylo dosaženo efektivního vývoje s úspěšným vytvořením bezpečné a rychlé webové aplikace.

Abstract

The .NET platform offers several frameworks, technologies and approaches for web applications development. A good survey of these technologies is necessary for selecting effective implementation technologies. The aim of the thesis was to create a conference information system as a web application on the .NET platform for the non-profit organization Zámecké návrší, p.o. including its deployment and operation in the production environment. Prior to the development, it was necessary to familiarize myself with the .NET platform and compare the frameworks for web development. The bachelor thesis presents the suitability of individual frameworks for different types of applications. Based on the .NET analysis, DotVVM framework was chosen to implement the conference information system. The development of application logic proceeded with the test-driven development method. Combining appropriate frameworks, technologies, and approaches has lead to effective development with the successful creation of a secure and fast web application.

Klíčová slova

Konferenční, informační, systém, Microsoft, .NET, ASP.NET, rámec, web, vývoj, webová aplikace, DotVVM, Riganti, GDPR, bezpečnost, test, testy řízený vývoj, Azure, MVVM, MVC, Razor, Web Forms, Web Pages, SPA, Web API

Keywords

Conference, information, system, Microsoft, .NET, ASP.NET, framework, web, develepment, web application, DotVVM, Riganti, GDPR, security, test, test-driven development, Azure, MVVM, MVC, Razor, Web Forms, Web Pages, SPA, Web API

Citace

BEDNÁŘ, Martin. *Konferenční informační systém.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Konferenční informační systém

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Další informace mi poskytl Martin Dybal. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Bednář
14. května 2018

Poděkování

Rád bych poděkoval vedoucímu své práce Ing. Zbyňku Křivkovi, Ph.D. za veškerý čas, který mi věnoval, za obětavost, s jakou vždy poskytl možnost konzultace a za jeho vstřícnost, s jakou vedl moji bakalářskou práci. Velké poděkování patří odbornému konzultantovi Martinu Dybalovi za odborné rady a předávání cenných zkušeností nabytých při své profesi softwarového architekta a vedoucího týmu programátorů.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Cílem této bakalářské práce je vytvořit konferenční informační systém jako webovou aplikaci na platformě .NET pro neziskovou organizaci Zámecké návrší, p.o. Konferenční systém je určen pro festival vzdělávání Nakopněte svoji školu, který je touto organizací pořádán v Litomyšli.

U systému se očekává reálné nasazení u zákazníka a také správa osobních údajů účastníků konference. Tím vzniká nutnost vyřešit problematiku zabezpečení aplikace a bezpečně pracovat v konferenčním systému s osobními údaji nejen na základě nové legislativy GDPR. Výsledná aplikace by měla splňovat všechny požadavky, které jsou aktuálně doporučeny pro zvýšení bezpečnosti webových aplikací. Jedním z takových požadavků je šifrování dat, a to nejen při komunikaci mezi klientskou a serverovou částí aplikace, kde je doporučené využívat protokol HTTPS umožňující zabezpečenou komunikaci, ale také šifrování databáze, které chrání dlouhodobě uchovávaná data proti jejich zneužití při možném útoku.

Pro efektivní vývoj a kvalitní výsledné řešení provedu analýzu webových rámců nad platformou .NET, z nichž bude vybráno konkrétní technické řešení dále využité při implementaci aplikace. V této oblasti práce se zaměří na analýzu zejména rámce DotVVM, který je z webových rámců nad platformou .NET jeden z nejmladších, za účelem zjištění jeho konkurenceschopnosti. Při analýze webových rámců se zaměřím i na otázku, pro který typ aplikace je daná technologie nejvhodnější a pro které typy aplikací naopak nemusí poskytovat vhodné nástroje. Pro podrobné proniknutí do znalosti webových rámců nad platformou .NET bude na základě každého rámce vytvořena malá ukázková aplikace a při analýze zdrojových kódů budou porovnány rozdílné principy funkčnosti. Rámce budou zasazeny při svém představení do kontextu historického vývoje celé platformy .NET s popsáním vazeb, která technologie navázala při svém vzniku na některou z dřívějších.

První část praktické práce cílí na analýzu požadavků od zákazníka. Na základě osobních setkání a konzultace s ředitelem Zámeckého návrší, p.o. bude provedeno podrobné zjištění potřeb zadavatele. Výstupem této části se rozumí sepsaná specifikace, která poslouží jako podklad při tvorbě návrhu konferenčního systému.

Návrh webové aplikace vytvoří dostatečný technický popis celého systému, na základě něhož bude možné zahájit implementaci. Návrh by se měl zabývat otázkou architektury a rozdělení aplikace do vrstev. Jelikož je požadován webový vývoj, očekává se vytvoření architektury, která odpovídá tomuto typu aplikace. Dále je třeba zvážit návrh databáze, který specifikuje schéma pro ukládání dat.

Ve své práci si kladu za cíl v implementační části ukázal a zdůvodnil výhody využití moderních webových rámců v kombinaci s testy řízeným vývojem. Tento přístup k vývoji softwaru nastuduji v teoretické fázi své práce a následně tento přístup využiji v části prak-

tické. Při implementaci se vyžaduje zachovávání jednotné kultury kódu s využitím pravidel pro čistý kód. Tento aspekt vývoje odráží potřebu možného rozšíření aplikace v budoucnu. Při implementaci je nutné dbát na všechna bezpečnostní omezení a zvolit vhodné postupy a nástroje k dosažení požadavků na bezpečnost konferenčního systému.

Při implementaci bude zohledněna také potřeba efektivního a časově nenáročného nasazení výsledného konferenčního systému do produkčního prostředí. Nasazení by měl předcházet výběr hostovacího prostředí a jeho následné otestování s cílem zhodnotit, zda vyhovuje potřebám zákazníka. Při hodnocení je důležitým aspektem množství zpracovaných požadavků za jednotku času a také průměrná doba odezvy.

Po vytvoření výsledného konferenčního systému budou vyhodnoceny návrhy na další možný rozvoj aplikace. Vývoj systému bude zakončen nasazením do produkčního prostředí a připravením údajů v něm na další ročník festivalu vzdělání. Podnětné nápady na další vývoj aplikace lze očekávat po analýze dat, která budou výstupem monitoringu aplikace při provozu v reálném prostředí na dalším ročníku festivalu vzdělání v Litomyšli.

Kapitola 2

Tvorba webových aplikací na platformě .NET

Tato kapitola nejdříve stručně představuje platformu .NET. Následně se zaměřuje výhradně na vývoj webových aplikací na této platformě.

Cílem aktuální kapitoly bylo zjistit, jaké prostředky (tj. například rámce) nabízí platforma .NET programátorovi k vytvoření webové aplikace a vzájemné porovnání těchto prostředků.

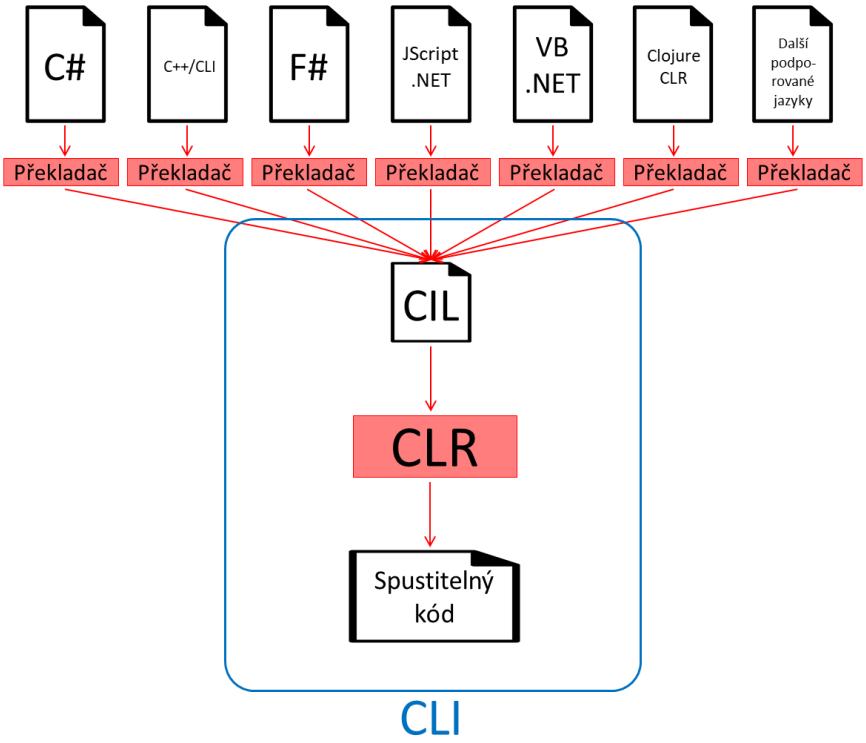
2.1 Platforma .NET

„Microsoft .NET“ je platforma pro vytváření a provozování aplikací. Jejími základními komponentami jsou společný jazykový běhový modul („Common Language Runtime“ – zkráceně CLR) a knihovna tříd rámce .NET („Framework Class Library“ – zkráceně FCL). [62]

CLR je virtuální běhové prostředí – jádro celého .NET, které překládá a spouští kód a poskytuje služby usnadňující proces vývoje. [59]

CLR ve spojení se společným mezikódem („Common Intermediate Language“ – zkráceně CIL) vytváří větší celek nazývaný „Common Language Infrastructure“ – zkráceně CLI. [16]

Funkci a význam CLI znázorňuje obrázek 2.1.



Obrázek 2.1: Common Language Infrastructure

Díky CLI je .NET jazykově nezávislý. Pro vývojáře to znamená, že může vyvíjet na platformě .NET v libovolném jazyku, který splňuje specifikaci společného zprostředkovacího jazyku („Common Language Specification“ – zkráceně CLS). Programovací jazyky je možné i kombinovat. Lze tak využít ve vlastní aplikaci jakoukoliv již naprogramovanou komponentu splňující CLS nehledě na její implementační jazyk. [30]

FCL neboli knihovna tříd rámce .NET poskytuje přístup k funkcionalitám operačního systému a zároveň zrychluje a optimalizuje vývojový proces. [58] Z názvu FCL lze vyčíst, že se jedná o knihovnu tříd. Není to však úplně přesné označení; daleko přesnějším označením by byl pojem „knihovna typů“, jelikož FCL může obsahovat cokoliv z následujícího: [62]

- třídy
- struktury
- rozhraní
- výčty
- delegáty

Jednou větou by bylo možné platformu .NET charakterizovat jako vrstvu vloženou mezi operační systém Windows a aplikace, která poskytuje soubor technologií potřebných k vývoji i spouštění těchto aplikací. [68]

2.1.1 Vznik a historie prostředí .NET

Prostředí .NET byl odborné veřejnosti představena poprvé v roce 2001. [82] Po rozsáhlém veřejném beta-testování, do něhož byly zapojeny desítky tisíc vývojářů, došlo v lednu roku 2002 ke zveřejnění platformy .NET. [14]

Platforma .NET vznikla na základě několika potřeb, které začaly v tehdejším vývoji aplikací vznikat nebo se výrazně projevovat a které bylo nutné řešit. Byly jimi zejména:

- Aplikace vyžadovaly skladbu z komponent, které často byly programované v různých jazycích a pro různá prostředí. Kvůli nekompatibilitě je bylo nutné přepisovat. .NET toto řeší vytvořením CLR, které bylo představeno v nadřazené kapitole. CLR zajistilo společné běhové prostředí pro různé jazyky podporované v .NET, a to klidně i současně na jednom projektu. [14]
- Do prioritního zájmu programátorů i uživatelů se začaly dostávat webové aplikace. Platforma .NET, respektive dále zmíněný a představený ASP.NET, nabídl nástroje pro snazší vývoj moderních webových aplikací. [82]
- V souvislosti s existencí různých hardwarových platforem a operačních systémů se ukazuje potřeba jednotného běhového prostředí či virtuálního stroje, které bude vytvářet univerzální prostředí pro aplikace. Na platformě .NET je program nezávislý na hardwaru a na operačním systému, jediné, co vyžaduje, je prostředí .NET. [82]

Ve skutečnosti se posledního zmiňovaného cíle — univerzálního prostředí pro aplikace — podařilo dosáhnout až s příchodem platformy .NET Core, jejíž první ostrá verze byla uvedena v červenci 2016. [23] Do té doby .NET sice představoval veřejný standard, který mohl fungovat na jakémkoliv systému, ale reálně Microsoft podporoval pouze operační systém Windows. [82]

.NET Core přinesl převratné novinky v přenositelnosti aplikací mezi různými operačními systémy. Velmi důležité pro komunitu vývojářů bylo a je, že se jedná o otevřený zdrojový kód. Zdrojový kód je zveřejněn na GitHub. Široká odborná veřejnost se tak mohla zapojit do oprav chyb a výkonnostních optimalizací. .NET Core je ale hlavně již od začátku navržen tak, aby jej bylo možné provozovat na Windows, Linux i MacOS (či OS X). Podporovaná jsou i zařízení s procesory ARM (např. Raspberry Pi). [23]

2.2 Typy aplikací na platformě .NET

Na platformě .NET můžeme vyvíjet mnoho různých druhů aplikací. Níže jsou vypsány nejčastější a nejvyužívanější typy.

Za zmínku stojí určitě klasické konzolové aplikace, které stále mají své nezastupitelné místo. Ty jsou na platformě .NET vyvíjeny s využitím třídy *System.Console*, která umožňuje číst a zapisovat znaky do konzole, kterou je v operačním systému Windows typicky program *cmd.exe*. [57]

Dále formulářové aplikace pro Windows, kde je uživatelské rozhraní vytvořeno pomocí knihovny *Windows.Forms* – interně využívající *Microsoft Win32 API*. [83]

Velké pozornosti se v posledních letech dostává aplikacím pro mobilní zařízení a univerzálním nebo přenositelným aplikacím. Například pomocí nástroje Xamarin mohou vývojáři tvořit kód pro zařízení, který je přenositelný mezi platformami Android, iOS a Windows. [10]

Dalším odvětvím jsou webové služby, které jsou často využívány pro přístup k nějakému datovému zdroji. [51] Nejedná se o plnohodnotné webové aplikace, ale opravdu pouze

o služby, které je možné vzdáleně volat pomocí protokolů HTTP a SOAP s využitím zpráv založených na XML. [15]

Poslední zmínka patří webovým aplikacím, na které se zaměří následující sekce.

2.3 Webový vývoj v .NET

V předminulé kapitole byla zmíněna potřeba vývoje webových aplikací. Před rokem 2002, kdy byla představena platforma .NET, existovala na poli webových technologií Microsoft pouze nedostačující skriptovací platforma „Active Server Pages“ – zkráceně ASP. [62]

Jako součást .NET byl v únoru 2002 představen i rámec ASP.NET, který slouží k vývoji webových aplikací na platformě .NET.

Webový rámec ASP.NET poskytuje sadu knihoven řešících mnoho základních problémů, s kterými se vývojáři setkávají v souvislosti s vývojem webových aplikací: bezpečnost, autentifikace uživatele, správa databází, správa formulářů atd. [96]

V názvu ASP.NET lze najít odkaz na předchozí webovou technologii ASP. Nejedná se však pouze o novou verzi klasického ASP. Rámec ASP.NET byl vytvořen od základu nově jako součást .NET. [15] Vznikla tak moderní infrastruktura pro vývoj webových aplikací, která je stále aktuální a hojně používá až do dnešní doby. Dle serveru BuiltWith.com bylo zastoupení rámce ASP.NET v lednu 2018 ve vzorku 10 000 nejvytíženějších webů (dle množství provozu na webu) procentuálně 21,8%. [69]

ASP i ASP.NET spojovala myšlenka přenést výpočetní zatížení na stranu serveru. Tedy umožnit běh klientské aplikace i na výkonnostně slabším zařízení, přes které uživatel webovou aplikaci obsluhuje. [62] Princip fungování je založen na vytvoření zdrojového kódu stránky již na straně serveru a pouze jeho odeslání do klientské aplikace. Na straně serveru je tak možné provést často náročné výpočty, programové konstrukce neznámé pro klientskou aplikaci, dynamické dosazení hodnot apod.

Aplikace je tak vysoce kompatibilní, jelikož k jejímu běhu na straně klienta stačí pouze webový prohlížeč.

Jestliže na straně klienta může být téměř jakékoliv prostředí, na straně serveru je omezení daleko přísnější. Webové aplikace využívající ASP.NET lze spouštět pouze na webovém serveru Microsoft IIS (Internetová Informační Služba, Internet Information Services) nebo na serveru UltiDev Web Server Pro, který nahradil starší server Cassini. [82] [36]

Od uvedení ASP.NET v tomto rámci vznikly a začaly převládat následující programátorské modely: [3]

- ASP.NET Web Forms
- ASP.NET MVC
- ASP.NET Web Pages
- ASP.NET Single Page Application
- ASP.NET Web API v kombinaci s „front-end“ knihovnami
- DotVVM

Všechny modely slouží k tvorbě webových aplikací na platformě .NET a lze jimi dosáhnout stejných nebo velmi podobných výsledků. Liší se však přístupem při programování. Každý přístup využívá jiné prostředky, které platforma .NET programátorovi nabízí.

V následujících podkapitolách jsou jednotlivé technologie a přístupy představeny. Ke každé technologii byla vytvořena malá aplikace demonstrující způsob vývoje a princip funkčnosti. Ukázkové aplikace v základu zobrazuji data z databáze. Na tomto případu užití je vidět využití všech vrstev aplikace od té datové až po zobrazení. V některých technologiích je přidáno do aplikace i textové pole pro vstup od uživatele, aby bylo demonstrováno i zpracování uživatelského vstupu. Kompletní zdrojový kód ukázkových aplikací je součástí práce a nachází se na přiloženém CD.

Toto seznámení se s problematikou bylo potřebným krokem před vybráním vhodných technických prostředků pro implementaci Konferenčního systému.

2.3.1 ASP.NET Web Forms

Webová aplikace založená na ASP.NET Web Forms se skládá ze dvou částí. První částí jsou webové stránky představované instancemi třídy *System.Web.UI.WebControls.Page*, jejichž obsah je tvořen běžným HTML kódem, do něhož jsou vloženy webové komponenty (tlačítko, vstupní pole, přepínač apod.) pomocí speciálního kódu definovaného v dokumentaci ASP.NET Web Forms. Druhou částí jsou funkce (kód na pozadí stránky), které jsou provedeny po zaznamenání příslušné události na dané ASP.NET komponentě. Tyto funkce většinou zpracovávají odezvu uživatele.

Základním principem tohoto přístupu je vytváření komponent na webových stránkách a navazování událostí k těmto komponentám. [28]

Webové komponenty jsou definovány v prostoru jmen *System.Web.UI.HtmlControls* – zde najdeme starší komponenty, které jsou zahrnuty zejména kvůli zpětné kompatibilitě – a *System.Web.UI.WebControls*. [82]

Níže je ukázka webové stránky, kde jsou využité komponenty *asp:button* a *asp:Label*. Ještě před začátkem HTML kódu je definován skript, který je následně volán při kliknutí na tlačítko s popiskem *Say Hello* a způsobí vypsání textu „Hello world!“ do komponenty *asp:Label* s id *greeting*.

```
1  <%@ Page Language="C#" %>
2  <script runat="server">
3      void sayHello(object sender, EventArgs e){
4          greeting.Text = "Hello world!";
5      }
6
7  </script>
8  <html>
9  <head>
10 <title>Hello World ASP.NET</title>
11 </head>
12 <body>
13     <form runat="server">
14         <asp:button id="sayhello" onclick="sayHello" runat="server"
15                 text="Say Hello"></asp:button>
16         <asp:Label id="greeting" runat="server"></asp:Label>
17     </form>
18 </body>
19 </html>
```

V běžných projektech dochází oproti ukázce k rozdělení vzhledu a logiky stránky do dvou samostatných souborů. Velkým ulehčením při tvorbě reálných aplikací je také využití náhledu „Designer“, přes který lze na webovou stránku komponenty umísťovat pomocí grafického rozhraní a principu „Drag-and-Drop“.

Základní vlastností formulářových komponent ASP.NET Web Forms je jejich zpracování na straně serveru. Když prohlížeč zašle požadavek na stránku, rámec ASP.NET provede potřebné skripty, poté vygeneruje HTML kód pro každou komponentu, sestaví HTML kód celé webové stránky a ten poté odešle do prohlížeče, který stránku vyrenderuje. [3]

Mnoho aplikací vyžaduje uchovávání určitých dat za svého běhu – například takzvaný stav aplikace. Zatímco v desktopových aplikacích je možné snadno uložit stav aplikace, ve webových aplikacích přichází komplikace na úrovni samotného komunikačního protokolu HTTP, který je bezstavový. Základní komunikace může vypadat následovně. Klient naváže spojení se serverem, odešle příkaz (například požadavek na stránku), server odpoví (pošle požadovanou stránku v odpovědi) a ukončí spojení. Celá transakce je následně zapomenuta a pokud klient za určitý čas pošle další požadavek, tak v základní HTTP komunikaci neexistuje způsob, jak se odkázat na předchozí komunikaci a navázat na ni. Potřeba uchovávat stav a vědět o předchozí komunikaci nastává například, když chceme sledovat chování uživatele nebo když chceme navázat na předchozí akce uživatele (například nabídnout v eshopu relevantní produkty či reklamu na základě předchozího chování). [79]

Pokud potřebujeme uchovávat stav aplikace, můžeme v ASP.NET Web Forms využít:

- „Cookies“ – malé množství informací ukládané na straně klienta. Tyto informace se přenášení s každým dalším požadavkem. [5]
- „Sessions“ – data jsou uchovávána na straně serveru a v „cookies“ nebo v URL se přenáší pouze identifikátor konkrétní „Session“.
- „ViewState“ – veškerá data jsou ukládána v zašifrované a komprimované podobě do skrytého formulářového pole __VIEWSTATE. Do „ViewState“ může programátor uložit libovolnou serializovanou hodnotu a při odeslání stránky (anglicky „postback“) ji ve „ViewState“ opět najde a může ji použít. [80]

ASP.NET Web Forms vznikly jako snaha přenést principy z vývoje formulářových / okenních aplikací pro Windows v .NET Win Forms na web. v době vzniku této diplomové práce je však už ASP.NET Web Forms považován za starší koncept, který se příliš neuplatňuje v nových projektech. [96] [17] [24] Podporu však stále má a společnost Microsoft již při představení ASP.NET MVC 1.0 potvrdila, že ASP.NET MVC (novější koncept, který je představen v následující kapitole) nemá být náhradou za ASP.NET Web Forms. [63] A že se neblíží ukončení jeho podpory naznačuje i to, že na základě ASP.NET Web Forms je vytvořena a stále funguje část dokumentace společnosti Microsoft. [41] Avšak sami programátoři potvrzují, že při vytváření nových projektů rozhodně převládá ASP.NET MVC oproti ASP.NET Web Forms. [78] [32] [4] [24]

Obecný posun vpřed v přístupu k vývoji softwaru se jasně ukazuje i v rozdílech mezi ASP.NET Web Forms a novějším konceptem ASP.NET MVC. V projektu založeném na ASP.NET Web Forms je velmi obtížné až skoro nemožné využít automatické testy a s tím i testy řízený vývoj (anglicky „Test-Driven Development“). Zatímco v projektu založeném na ASP.NET MVC je pro automatické testy velká podpora – dokonce je možné vytvořit samostatný projekt na testy ihned při vytvoření nového řešení. [56]

Já osobně vnímám ASP.NET Web Forms jako velmi kvalitní technologii, která brilantně reagovala na situaci v roce 2002. Při svém uvedení dokázal ASP.NET Web Forms výrazným

způsobem ulehčit vývoj webových aplikací. Jeho místo zůstalo však v letech 2002 až 2009, kdy byl představen nový rámec ASP.NET MVC.

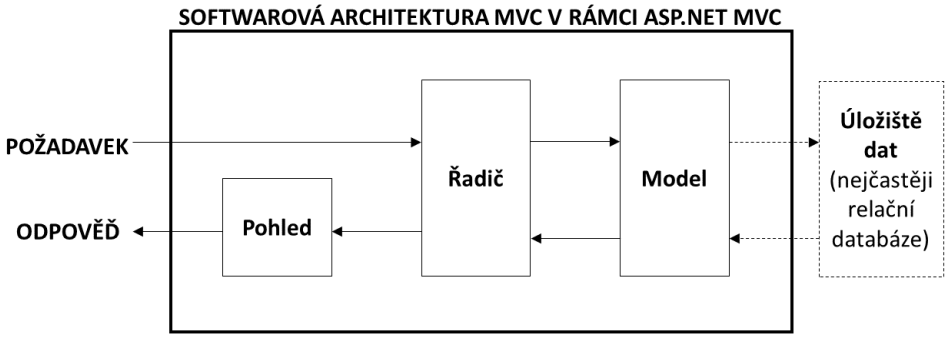
2.3.2 ASP.NET MVC

ASP.NET MVC je rámec pro webový vývoj vydaný roku 2009 společností Microsoft. Rámec používá v projektech architektonický vzor Model-View-Controller (MVC). Tento architektonický vzor je pro daný rámec natolik typický, že se jeho zkratka promítla i do názvu samotného rámce. [20]

Architektura MVC je krátce představena v následující podkapitole.

V roce 2009, kdy byl představen rámec ASP.NET MVC byl architektonický vzor MVC již dobře známý a rozhodně se nedá říct, že by se jednalo o nějaký nový přístup. Architektonický vzor MVC se datuje až do roku 1978, kdy byl poprvé využíván na Smalltalk projektech v Xerox PARC. Ten vzor si získal velkou pozornost při vývoji webových aplikací z důvodu práce s třemi komponentami, které mohou při webovém vývoji odpovídat jednotlivým technologiím a vrstvám aplikace (např. uchovávání dat a přístup k databázi, HTML, spustitelný kód vykonávající logiku aplikací). [20]

Implementaci architektury MVC v ASP.NET znázorňuje obrázek 2.2.



Obrázek 2.2: Implementace MVC v ASP.NET

Stejně jako v předchozí kapitole o ASP.NET WebForms i zde uvádím kód jednoduché aplikace – tentokrát vytvořenou v rámci ASP.NET MVC. Uvedení zdrojového kódu je zejména za účelem porovnání. V ukázce je názorně vidět, že v ASP.NET MVC došlo ke striktnějšímu rozdělení zdrojového kódu dle architektury MVC. Kód se stává přehlednější a složité aplikace jsou lépe udržovatelné a rozšiřovatelné. Ukázková aplikace po uživatelově kliknutí na tlačítko s popiskem *Say Hello* vypíše pozdrav zadaný do textového vstupního pole na hlavní stránku webové aplikace.

Pohled – soubor *Index.cshtml*:

```
1 @model WebApplicationMVC.Models.Greeting
2 @{
3     ViewBag.Title = "Index";
4 }
5 @using (Html.BeginForm())
6 {
```

```
    <div class="form-horizontal">
        <div class="form-group">
            Pozadovany pozdrav:
            <div>
                @Html.EditorFor(model => model.InputGreeting)
            </div>
        </div>

        <div class="form-group">
            <div>
                <input type="submit" value="Say Hello" class="btn btn-
                    default" />
            </div>
        </div>
    </div>

    <div>
        <center><p style="font-size:3em;">@Model.OutputGreeting</p></center>
    </div>
}
```

Řadič – soubor *HomeController.cs*:

```
...
public class HomeController : Controller
{
    public ActionResult Index()
    {
        Greeting greeting = new Greeting();
        return View(greeting);
    }

    [HttpPost]
    public ActionResult Index(Greeting greeting)
    {
        greeting.OutputGreeting = greeting.InputGreeting;
        return View(greeting);
    }
}
...
```

Model - soubor *Greeting.cs*:

```
...
public class Greeting
{
    public String InputGreeting { get; set; }
    public String OutputGreeting { get; set; }
}
...
```

V ukázce zdrojových kódu si je možné všimnout, že pohled je popsán značkovacím jazykem HTML, který je obohacen o provázání dat z Modelu. Toto provázání je ve zdrojovém kódu pohledu uvozeno znakem @. Řadič obsahuje dvě metody se stejným jménem, ale s různým počtem parametrů. Zatímco metoda bez parametrů je volána vždy při požadavku na nové načtení webové stránky, metoda s jedním parametrem a označením *[HttpPost]* je volána v případě odeslání formuláře (v případě metody post). Model uchovává stav webové stránky. Při odeslání formuláře jsou do něj uložena navázaná data a při načtení stránky jsou z něho hodnoty načteny, aby pohled zobrazil jejich aktuální hodnoty.

V bodech níže jsou shrnuté výhody ASP.NET MVC:

- Rozdělení zdrojového kódu dle architektury MVC přináší kromě lepší rozšiřitelnosti aplikace a přehlednosti zdrojového kódu také přívětivější prostředí pro spolupráci více vývojářů. Každý vývojář může pracovat v určitých zdrojových souborech (například pouze na Pohledu) a díky tomu nedochází ke konfliktům při spojování zdrojových kódů od několika vývojářů. Spoluprací více vývojářů se může zrychlit vývoj cílové aplikace. [38]

- Další nespornou výhodou je přímá podpora testů a testy řízeného vývoje v rámci. Testy jsou potřebnou částí softwarového řešení zejména v dlouhodobě vyvíjených a udržovaných projektech, kde změna nebo přidání funkcionality nesmí ovlivnit funkčnost stávajícího řešení. [88]

- V porovnání s ASP.NET Web Forms přinesl ASP.NET MVC jednodušší a pro uživatele čitelnější URL webových stránek. Zatímco aplikace vytvořená v ASP.NET WinForms se dala snadno rozpoznat pomocí složité URL a příponou *.aspx* u jednotlivých výsledných webových stránek, ASP.NET MVC nejen zjednodušuje URL, ale také nepoužívá charakteristickou koncovku. Nejlépe je rozdíl patrný v praktických ukázkách dále. URL v ASP.NET WebForms může vypadat například takto: $http://domena.moje/App\_v2/User/Page.aspx?action=show\%20prop\&prop\_id=123$. Oproti tomu v ASP.NET MVC došlo ke zjednodušení URL: $http://domena.moje/deployment/Csharp$. [20]

ASP.NET MVC měl a stále má nesporný přínos pro webový vývoj. Přináší však také nevýhody (některé pouze subjektivní), mezi které patří například:

- Mezi zápornou vlastnost subjektivního charakteru patří absence náhledu designu ve Visual Studiu (je však možné „náhled“ otevřít přímo ve webovém prohlížeči a při změně zdrojového kódu vzhledu není potřeba ani znovu aplikaci zkompilovat; pro změnu designu stačí pouze aktualizovat stránku). [38]

- Další nevýhodou, kterou už nelze označit pouze za subjektivní je potřeba znalosti většího počtu rozdílných webových technologií. Například pro vytvoření dynamických webových stránek nebo složitějších formulářů potřebujeme kromě znalosti HTML, CSS a C# také JavaScript a různé jeho knihovny nebo rámce – např. AJAX, AngularJS, React.js či jQuery. [71] V porovnání s dále zmíněným DotVVM se jedná o výrazně složitější kombinaci technologií. V DotVVM stačí k dosažení stejného výsledku pouze znalost HTML, CSS a C#. [67]

- Vývoj v ASP.NET MVC vyžaduje pochopení architektury MVC a celého konceptu rámce, který v porovnání například s WebForms nebo s DotVVM je složitější na porozumění. Složitost použití má poté vliv na rychlost vývoje nového projektu v daném

rámci případně ochotu programátorů a firem přejít na tento modernější rámec ze starších známých technologií. [34]

Rámec ASP.MVC nabízí vše, co vývojář moderních webových aplikací v dnešní době potřebuje – ať už to jsou testy včleněné do řešení či správa uživatelů, kterou je možné integrovat do aplikaci již při jejím vytvoření. V porovnání s jinými přístupy na platformě ASP.NET (konkrétně ASP.NET Web Forms a DotVVM) nemusí být vývoj tak rychlý a zpočátku snadný.

Architektonický vzor Model-View-Controller

Architektura Model-View-Controller (MVC), která je implementována v ASP.NET MVC, se vyznačuje rozdělením uživatelského rozhraní, řídicí logiky a přístupu k datům do tří samostatných komponent: [40]

- Model („Model“) obsahuje data a doménovou logiku, což jsou akce nad těmito daty. Model se celkově stará o držení a zpracovávání dat. Předává aktuální stav dat (typicky do komponenty Pohled) a přijímá požadavky na aktualizaci dat (typicky od komponenty Řadič). V komponentě Model je možné řešit například problematiku uchovávání dat – zda budou dočasně uložena v seznamu, v poli nebo struktuře; a dále například naprogramování metody pro přidání nového prvku.
- Pohled („View“) zajišťuje interaktivní reprezentaci dat v uživatelském rozhraní aplikace. Cílem je prezentovat data z komponenty Model uživatelsky přívětivým způsobem. V komponentě Pohled je možné řešit například problematiku zobrazení datumu, aby byl přizpůsobený aktuální lokalizaci, nebo responzivitu aplikace.
- Řadič („Controller“) reaguje na události a zajišťuje změny v Modelu, případně i v Pohledu. Mohl by být označen jako řídící komponenta, která má za úkol provázání funkčnosti celé aplikace. V komponentě Řadič je možné řešit například zpracování kliku na tlačítko.

Nejen v architektuře MVC platí princip, že při úpravě jedné z komponent (Model, Pohled, Řadič) se tato modifikace ideálně neprojeví v ostatních komponentách. Například, když v komponentě Model změníme způsob uchovávání dat, v ostatních dvou komponentách (Pohled a Řadič) by nemělo dojít k žádné změně a potřebě úprav. [7]

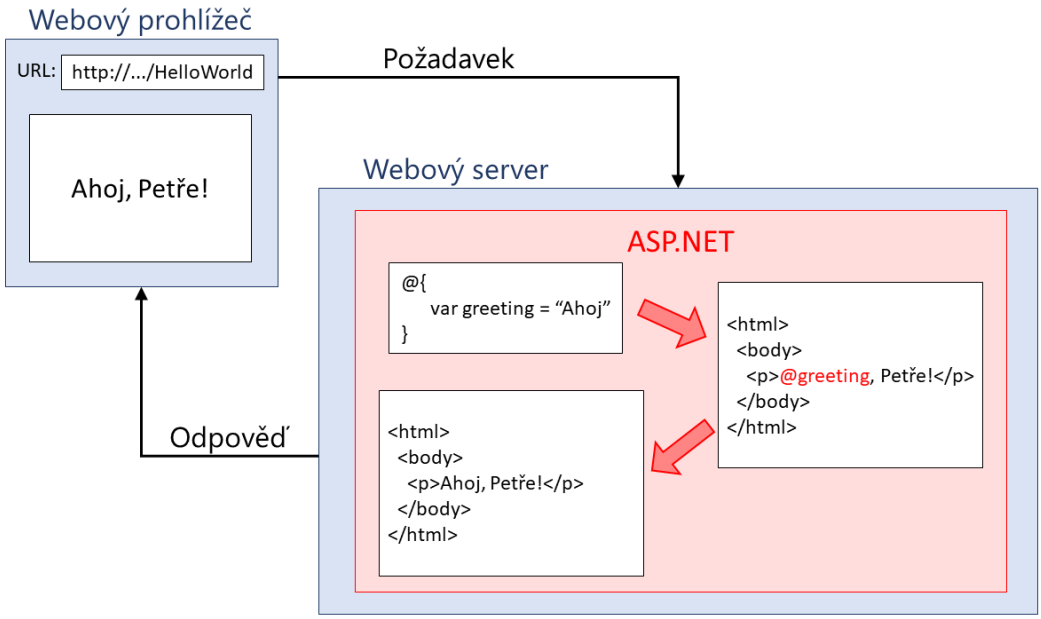
2.3.3 ASP.NET Web Pages

ASP.NET Web Pages jsou webové stránky vytvořené pomocí značkovacího jazyku HTML a kódu syntaxe Razor. HTML kód definuje zobrazení dat (jejich umístění na stránce, formátování apod.) a kód syntaxe Razor popisuje práci s daty a operaci nad nimi. [39]

Razor je jednoduchá syntaxe založená na programovacím jazyku C#, Kód syntaxe Razor je na webových stránkách prováděn na straně serveru a běží na serveru před odesláním stránky do prohlížeče. Tento kód může dynamicky vytvářet klientský obsah – tedy generovat značky HTML či samotný obsah za běhu aplikace a poté jej odesílat do prohlížeče spolu se statickým kódem HTML. [19]

Jak probíhá zpracování požadavku na webovém serveru popisuje obrázek 2.3.

ASP.NET Web Pages jsou zatím nejjednodušším konceptem z dosud představených. Jejich architektura není nijak členěna, jedna webová stránka odpovídá jednomu zdrojovému souboru. Zároveň tento koncept je velmi snadný na naučení a rychlé pochopení. Velkou



Obrázek 2.3: Zpracování požadavku na server ASP.NET se zpracováním kódu syntaxe Razor

výhodou může být také šablona, ze které se vytváří ve Visual Studio nový projekt. Ta obsahuje mnoho kompletních předvytvořených stránek, které běžná jednoduchá webová aplikace potřebuje (Hlavní stránka, O nás, Kontakt, Registrace, Přihlášení, Potvrzení účtu, Zapomenuté heslo, Odhlášení, Resetování hesla, ad.).

ASP.NET Web Pages jsou ideálním nástrojem pro velmi rychlé vytvoření jednoduché webové aplikace. Vzhledem k jednoduchosti a rychlosti vytvoření i samotného vývoje webové aplikace mohou nabízet konkurence schopnou alternativu k ASP.NET Web Forms. Velkým rozdílem mezi těmito dvěma přístupy, který nemusí být viditelný na první pohled, je způsob tvorby výsledného HTML obsahu webové stránky. V ASP.NET Web Forms jsou jednotlivé webové kontrolky popsány pomocí kódu, který je prováděn na serveru. Výsledný HTML kód je kompletně generován na serveru při běhu serverové části aplikace. Naproti tomu v ASP.NET Web Pages je většina ovládacích prvků popsána v statickém HTML kódu a pouze funkcionalita je k těmto prvků přidána pomocí serverového kódu. [2]

Rozdíl mezi ASP.NET Web Forms a ASP.NET Web Pages si lze ukázat na kontrolce tlačítko. V ASP.NET Web Forms pro vytvoření tlačítka standardně napíšeme následující kód (nebo si ho necháme vygenerovat po vložení tlačítka v náhledu Designer):

<asp:Button ID="Button1"runat="server"Text="Button"/>

Vidíme, že se jedná o kód, který musí být na serveru přeložen do HTML a teprve poté je odeslán do klientského prohlížeče.

V ASP.NET Web Pages pro vytvoření tlačítka píše vývojář přímo výsledný HTML kód. Na straně serveru tak nedochází k překladu do HTML, pouze se řeší funkcionalita navázaná k tlačítku:

<input type="submit"value="Pozdravit"/>

V obou případech je výsledek na klientovi stejný nebo velmi podobný. Rozdíl je ve vývoji a zpracování zdrojového kódu. Zatímco v ASP.NET Web Forms jsou webové kontrolky psány v syntaxi, kterou definuje ASP.NET Web Forms (nebo jsou vkládány přes grafické rozhraní) a z nich je následně na serveru vytvořen výsledný HTML kód, v ASP.NET Web Pages se

programátor více přibližuje už při vývoji výslednému kódu, když kontrolky popisuje pomocí HTML, které je následně bez většího serverového zpracování zasíláno na klienta.

Tímto principem se ASP.NET Web Pages přiblížili daleko více vývoji statických webových stránek, k němuž stačí znalost jednoduchého HTML, nežli složitějšímu vývoji klientských desktopových aplikací. Z tohoto důvodu a protože je syntaxe jednoduchá, může být pro programátory jednodušší seznámit se s prostředím Razor a mohou jeho pomocí rychleji vytvářet webové stránky. [43]

Zdrojový kód ukázkové aplikace níže demonstruje užití ASP.NET Web Pages. Soubor se zdrojovým kódem webové stránky má příponu *.cshtml* (případně *.vbhtml*, pokud by byl projekt psaný v programovacím jazyku Visual Basic) Zdrojový kód se skládá z HTML značek a kódu syntaxe Razor, který je vždy uvozen znakem @. Ukázková aplikace čeká na zadání pozdravu. Po zadání pozdravu do vstupního textového pole se jménem *Greeting* a odeslání formuláře kliknutím na tlačítko *Pozdravit* dojde k vypsání pozdravu pro všechny uživatele v databázi v tabulce *Person*. Pro přístup k databázi jsou využívány metody ze třídy *WebMatrix.Data.Database*. Za zmínku stojí, že všechny vrstvy, které by se daly rozdělit dle zvoleného architektonického vzoru (pohled, řadič, logika aplikace, model, přístup k datům...) se nachází v jediném zdrojovém souboru. Zde je vidět velký rozdíl oproti ASP.NET MVC nebo dále zmíněném DotVVM, které jsou založeny na jasném rozdělení zdrojového kódu dle architektonického vzoru. V ukázce tak můžeme vidět, že zobrazení dat, logika aplikace i přístup k datům v databázi jsou v jediném zdrojovém souboru, jak to je pro ASP.NET Web Pages charakteristické.

Soubor *Default.cshtml*:

```
@{
    var db = Database.Open("Greeting");
    var selectQueryString = "SELECT * FROM Person ORDER BY Name";
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Pozdrav</title>
</head>
<body>
<div>
    <form method="post">
        Pozadovany pozdrav:
        <input type="text" name="Greeting" /><br /><br />
        <input type="submit" value="Pozdravit" />
    </form>
</div>
@{
    if (IsPost) {
        string greeting = Request["Greeting"];
        <div>
        @foreach (var row in db.Query(selectQueryString))
        {
```

```
25              <center><p style="font-size: 3em;">@greeting,@row.Name</p
26              </center>
27          }
28          </div>
29      }
30  }
31  </body>
32  </html>
```

ASP.NET Web Pages se výborně hodí pro tvorbu jednodušších webových aplikací, u kterých je požadavek na rychlé a snadné vytvoření. Zároveň mohou výborně posloužit i jako jednoduchá technologie pro naučení se práce s ASP.NET. Díky své jednoduchosti a rychlosti vývoje mohou být využity jako modernější alternativa k ASP.NET Web Forms. Naopak ASP.NET Web Pages se nedají doporučit pro tvorbu rozsáhlých, komplexních a dlouhodobě vyvíjených aplikací. U takového druhu vývoje by velmi brzy došlo k nepřehlednosti a zbytečnému kopírování zdrojového kódu.

2.3.4 ASP.NET Single Page Application

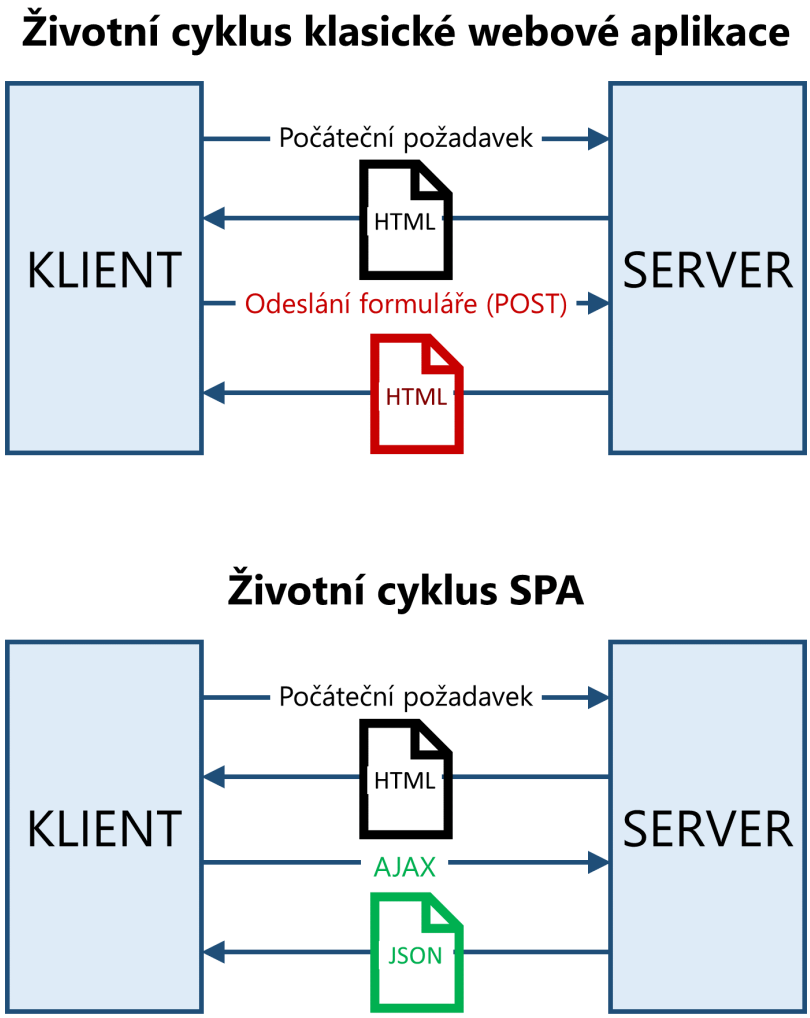
ASP.NET Single Page Application (ASP.NET SPA) je moderní koncept webových aplikací, který byl představen s vydáním 4. beta verze ASP.NET MVC v roce 2012. [54]

ASP.NET SPA přináší výrazné změny ve webovém vývoji. Již samotný název „Single Page Application“ by mohl vést k dojmu, že celá aplikace je pouze jediná webová stránka (jediný pohled). Toto však není přesné. Zatímco obecný koncept „Jednostránkový web“ (anglicky „Single Page Website“) opravdu znamená, že všechny informace (všechny pohledy) jsou zobrazeny na jediné webové stránce (s jediným URL) a mezi nimi se pohybuje pomocí skorolování nebo pomocí záložek (anglicky „bookmarks“), [72] v ASP.NET je tento koncept vnímán jinak. ASP.NET SPA přináší nový přístup, že celá aplikace je načtena do prohlížeče při jejím otevření (odtud název „Single Page Application“ – celá aplikace se chová jako jedna stránka, protože je načtena celá při prvním otevření). [91]

Nabízí se otázka, proč Microsoft zvolil tento netradiční přístup? K odpovědi dojdeme, když se vrátíme k jednomu z důvodů vzniku ASP.NET. Tato webová technologie vznikla proto, aby bylo možné vyvíjet a provozovat webové aplikace, které se budou svojí složitostí a komplexností blížit klasickým desktopovým aplikacím. [95] Tohoto cíle se velkou měrou podařilo dosáhnout i s jinými webovými technologiemi na platformě ASP.NET než je ASP.NET SPA. [28] Avšak jedna vlastnost webových aplikací je stále velmi odlišovala a v jisté míře stále odlišuje od klasických desktopových aplikací, které běží lokálně na počítači – a tím je doba odezvy a doba nutná k načtení další stránky (pohledu). ASP.NET SPA se tímto zabývá a svým řešením se ještě více přibližuje klasické desktopové aplikaci. Tím, že celá aplikace je načtena již při prvním přístupu, tak přechod mezi jednotlivými stránkami je okamžitý. Ve chvíli, kdy uživatel klikne v menu na odkaz na jinou stránku, tak neproběhne klasický požadavek na server, odpověď serveru a zpracování odpovědi, ale pouze změna pohledu, který již byl načten dříve, a případná aktualizace některých dat (například z databáze). Změny v datech aplikace se synchronizují se serverem v reálném čase prostřednictvím technologie AJAX. [89]

Jak v ASP.NET SPA funguje komunikace přes síť znázorňuje obrázek 2.4. Zatímco klasická webová aplikace (vytvořená například pomocí ASP.NET MVC) funguje na základě zasílání zdrojového kódu webové stránky a jeho zpracování prohlížečem poté, co klientská

aplikace (prohlížeč) zašle požadavek na novou stránku nebo odešle formulář, ASP.NET SPA pouze aktualizují data v již přednačtené stránce.



Obrázek 2.4: Porovnání životního cyklu klasické webové aplikace a životního cyklu ASP.NET SPA webové aplikace

Tím, že ASP.NET SPA jsou jedinou stránkou, která je celá načtena a uložena lokálně při prvním přístupu uživatele do aplikace, dochází k přenesení části zátěže ze serveru na klientskou část. Dochází tak k částečnému upozadění tradičního konceptu webových aplikací, který je založen na přenesení co největší možné zátěže na server a zobrazování dat v tenké klientské aplikaci. ASP.NET SPA naopak minimalizují server a kladou větší důraz na klientskou část. Vzhledem k výkonu moderních zařízení je však tento fakt do jisté míry zanedbatelný.

Příkladnou ukázkou SPA mohou být známí weboví e-mailoví klienti Gmail, Seznam Email nebo RoundCube. Na těchto webových aplikacích lze pozorovat charakteristické chování SPA:

- První načtení aplikace při jejím otevření v prohlížeči trvá delší dobu než u klasických webových aplikací.
- Pohyb v aplikaci po jejím načtení je rychlejší, než v klasické webové aplikaci (například přechod mezi jednotlivými složkami pošty).
- Při odkazování se na jednotlivé stránky jsou využívány záložky (anglicky „bookmarks“). Například *email.seznam.cz/#inbox* pro příchozí e-maily, *email.seznam.cz/#sent* pro odeslané e-maily nebo *email.seznam.cz/#drafts* pro rozepsané e-maily.
- Při zaslání požadavku na načtení nové stránky se načítají pouze nezbytná dynamická data. Vše ostatní zůstává již načteno a zobrazeno v prohlížeči. Nedochází tak například k neustálému překreslování hlavního menu.

Nezvyklý přístup v ASP.NET SPA čeká .NET vývojáře při tvorbě aplikace. Většinu zdrojového kódu napíše v jazyku JavaScript. Je to způsobeno přenesením větší části zátěže na stranu klienta, čemuž při programování odpovídají klientské skripty v jazyku JavaScript. Ve výchozím projektu Visual Studia je v šabloně pro ASP.NET SPA připraven javascriptový framework knockout.js. Kromě knockout.js je však možné použít například také AngularJS nebo backbone.js. [97]

Odlišnost přístupu k vývoji demonstruje část ukázkové aplikace níže. Aplikace se připojí k databázi a po svém načtení vypíše pozdravy, které jsou uloženy v databázi v tabulce *Greetings*. Data z databáze se získávají přes API (anglicky „Application Programming Interface“), které je navázáno na URL *api/Greeting*. Data jsou získávána v serializovaném formátu JSON po volání funkce *returnAllGreetings()* vytvořené v jazyku JavaScript. Pro vazbu mezi daty (anglicky „data-binding“) je využíván javascriptový framework knockout.js, který zjednoduší vkládání dat do HTML.

Serverová část aplikace je představována technologií ASP.NET MVC. Zejména důležité jsou v tomto případě řadiče, které zajišťují fungování API.

Pohled – soubor *_Greeting.cshtml*:

```
1  <!-- ko with: greeting -->
2  <div class="jumbotron">
3      <h1>Pozdravy</h1>
4
5      <table>
6          <tbody data-bind="foreach: greetings">
7          <tr>
8              <td>
9                  <span data-bind="text: choosengreeting"></span>,
10                 <span data-bind="text: name"></span>!
11             </td>
12         </tr>
13         </tbody>
14     </table>
15
16 </div>
17 <!-- /ko -->
```

„ViewModel“ – soubor *greeting.viewmodel.js*:

```
function Greeting(id, choosengreeting, name) {
    return {
        id: ko.observable(id),
        choosengreeting: ko.observable(choosengreeting),
        name: ko.observable(name)
    };
}

function GreetingViewModel(app, dataModel) {
    var self = this;

    // Greeting collection
    self.greetings = ko.observableArray([]);
    // Load all greetings
    self.returnAllGreetings = function () {
        $.getJSON("/api/Greeting", function (data) {
            var jsonData = data;
            for (var i = 0; i < jsonData.length; i++) {
                var greeting = new Greeting(jsonData[i].id, jsonData[i].
                    choosenGreeting, jsonData[i].name);
                self.greetings.push(greeting);
            }
        })
    };
    self.returnAllGreetings();
}

// Register ViewModel
app.addViewModel({
    name: "Greeting",
    bindingMemberName: "greeting",
    factory: GreetingViewModel
});
```

Řadič – soubor *GreetingController.cs*:

```
...
public class GreetingController : ApiController
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: api/Greeting
    public IQueryable<Greeting> GetGreetings()
    {
        return db.Greetings;
    }
...
```

Model – soubor *Greeting.cs*:

```
...
public class Greeting
{
    public int ID { get; set; }

    public string ChoosenGreeting { get; set; }

    public string Name { get; set; }
}
...
```

ASP.NET SPA jsou moderním konceptem, díky kterému lze vytvářet svižné a uživatelsky přívětivé aplikace. Při vývoji je třeba mít na paměti, že SPA načítají celou webovou aplikaci již při prvním přístupu uživatele, a tudíž by mohlo dojít k zabrání velké části paměti klienta, pokud by aplikace byla příliš rozsáhlá.

Z pohledu programátora patří vývoj v ASP.NET SPA k náročnějším přístupům. Kromě znalostí klasických technologií v ASP.NET (C#, EntityFramework, architektonický vzor MVC atd.) musí programátor ovládat na pokročilé úrovni ještě JavaScript (s knockout.js nebo obdobným rámcem) a práci přes REST API.

V ASP.NET SPA lze dosahovat velmi zajímavých a efektivních výsledných aplikací za cenu náročnějšího vývoje a přenesení větší části zátěže na stranu klientské aplikace.

2.3.5 ASP.NET Web API v kombinaci s „front-end“ knihovnami

Samotné ASP.NET Web API je technologie, která by patřila zařadit do kategorie „Webové služby“ (anglicky „Web Services“) spíše než do plnohodnotných webových aplikací. ASP.NET Web API slouží totiž pouze k vytvoření API ve vrchní vrstvě rámce .NET, přes které může s .NET aplikací komunikovat například webová stránka a může z něj načítat a získávat data. [90]

ASP.NET Web API řeší pouze logiku aplikace a vytvoření API, přes které bude k datům poskytovat přístup. Samotné zobrazení dat a celkově vytvoření pohledu v aplikaci zůstává čistě v rukou vývojáře a může zvolit libovolnou technologii. Nejčastěji se využívají různé JavaScript knihovny. [90]

Výraznou nevýhodou tohoto přístupu je nutnost napsat velké množství kódu v jazyku JavaScript, který nedělá nic jiného, než že vezme data z API a přenese je do pohledu. [22]

Naopak velkou výhodu ASP.NET Web API přinese vývojářům, kteří pracují na aplikaci určené pro mnoho různých platforem. Například, když aplikace ve výsledku bude vytvořena jako desktopová, webová a mobilní, tak je možné využít stejnou logiku aplikace, ke které se přistupuje právě pomoci API a vytvořit pouze zobrazení těchto získaných dat na konkrétním zařízení. Opět však bude potřeba napsat velké množství kódu v jazyku JavaScript, který pouze přenáší data do pohledu. [12]

V ukázkové webové aplikaci níže zprostředkovává API přístup k údajům v databázi v tabulce *Greetings*. K zobrazení dat je v této ukázce využita knihovna jQuery, která zašle AJAX („Asynchronous JavaScript and XML“) požadavek pro získání dat přes API, následně je zpracuje a nakonec zobrazí. V ukázce je jako zdroj knihovny jQuery využit „Microsoft Ajax Content Delivery Network“ [1]. Při zaslání požadavku na API dojde na straně serveru k serializaci modelu do formátu JSON a jejich odeslání. Na webovou stránku

se vypíší hned při načtení pozdravy předdefinované v databázi.

Pohled – soubor *Index.html*:

```
...
<body>
    <div id="greetings"></div>

    <script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.0.3.min.js
        "></script>
    <script>
        var uri = 'api/Greeting';

        $(document).ready(function () {
            // Send an AJAX request
            $.getJSON(uri)
                .done(function (data) {
                    $.each(data,
                        function (key, item) {
                            $('<p>', { text: item.ChoosenGreeting + ', ' +
                                item.Name }, '</p>').appendTo($('#greetings'))
                                ;
                        });
                });
        });
    </script>
</body>
...
```

Řadič – soubor *GreetingController.cs*:

```
...
public class GreetingController : ApiController
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: api/Greeting
    public IQueryable<Greeting> GetGreetings()
    {
        return db.Greetings;
    }
...
```

Model – soubor *Greeting.cs*:

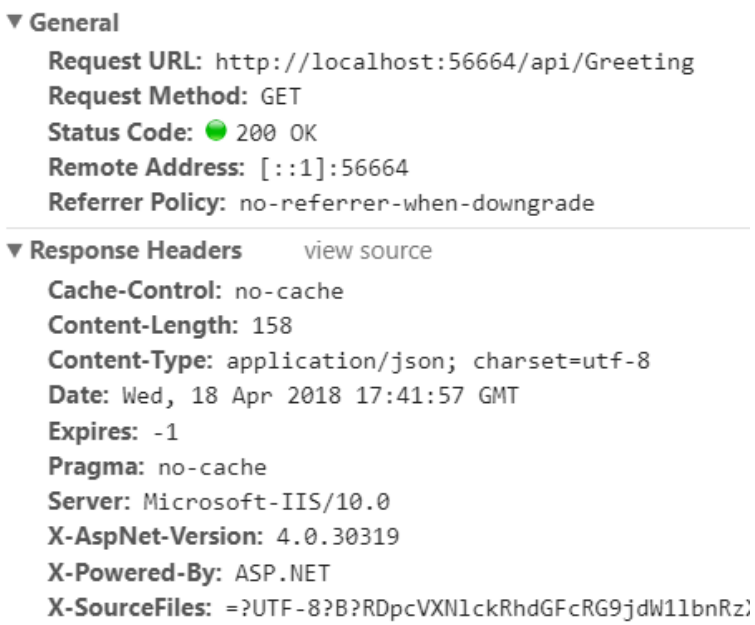
```
...
public class Greeting
{
    public int ID { get; set; }

```

```
6       public string ChoosenGreeting { get; set; }
7
8       public string Name { get; set; }
9   }
10  ...
```

Můžeme vidět, že tvorba aplikace v ASP.NET Web API připomíná svých přístupem a použitými programovacími jazyky ASP.NET Single Page Application. Na rozdíl od ASP.NET SPA nedochází při použití ASP.NET Web API k načtení celé webové aplikaci při jejím prvním otevření, ale komunikace probíhá klasickým způsobem, kdy při změně stránky je nová stránka získána pomocí HTTP požadavku ze serveru a po obdržení odpovědi od serveru je načtena do prohlížeče.

Ve spuštěné aplikaci můžeme sledovat komunikaci přes API například pomocí programu WireShark nebo můžeme využít přímo webový prohlížeč podporující sledování síťové komunikace (např. Google Chrome). [60] [6]

Ze zachycené komunikace na obrázku 2.5 můžeme vyčíst, že API je dostupné na URL */api/Greeting*. Server zasílá data serializovaná ve formátu JSON v kódování UTF-8 o délce 158 bytů. ASP.NET aplikace běží na webovém serveru IIS.



Obrázek 2.5: Síťová komunikace webové aplikace přes API (odpověď serveru)

ASP.NET Web API lze rozhodně zařadit do moderních přístupů k vývoji webových aplikací a nejen jich. Pokud však cílíme pouze na webovou aplikaci a nepotřebujeme cílit na mnoho různých platforem, chybí v ASP.NET Web API kompletně vrstva pro zpracování pohledu. Vše, co se týká pohledu si musí programátor vytvořit sám. Protikladem v této práci s pohledem může být ASP.NET MVC nebo v dále představený DotVVM, kde lze data vypisovat s využitím propojení dat mezi pohledem a modelem (anglicky „data-binding“). Pokud však vývojář potřebuje hlavně webovou službu (anglicky „web service“) je ASP.NET Web API ideálním řešením.

2.3.6 DotVVM

DotVVM je webový rámec založený na technologii ASP.NET. [31] Vyvíjí ho česká společnost Riganti s.r.o. První verze byla představena v roce 2015. [25] Rámec DotVVM je vyvíjen jako otevřený kód (anglicky „open-source“), který je přístupný na GitHubu firmy. [26]

DotVVM cílí na vývojáře rozsáhlých systémů vytvářených jako webové aplikace. Zejména na systémy, které jsou velkou částí tvořeny formuláři a zobrazením dat z odeslaných formulářů. Na tento případ užití je v rámci připraveno mnoho komponent, které lze velmi snadno použít a pouze jim předat potřebná data – například GridView, Button, CheckBox, FileUpload, TextBox, Repeater a další. [65] DotVVM však není jen sada hotových ovládacích prvků pro web. Rámec dále zpracovává komunikaci klient-server, lze v něm snadno nastavit validaci dat při odeslání formuláře nebo zajišťuje nástroje pro formátování dat (například data a času). [67]

Pro vytvoření aplikace v DotVVM stačí programátorovi pouze znalost HTML, CSS a C#. Rámec odstiňuje vývojáře od veškerého kódu v jazyku JavaScript. Pokud si zobrazíme výsledný zdrojový kód, který se posílá do prohlížeče na straně klienta, můžeme si všimnout, že rámec interně využívá knockout.js pro provázání dat (anglicky „data-binding“) – viz obrázek 2.6. Tímto se však programátor nemusí vůbec zabývat. Vývojáři stačí nastavit navázání dat mezi „View“ a „ViewModel“ pomocí speciální syntaxe DotVVM.

```
▼<p>
   "Počet zaregistrovaných účastníků: "
  ▼<b>
     <!-- ko text: TotalPeopleCount -->
     "152"
     <!-- /ko -->
   </b>
 </p>
```

Obrázek 2.6: Výsledný HTML kód demonstrující interní využití knockout.js v rámci DotVVM

Pokud by však programátor potřeboval vložit do stránky vlastní JavaScript kód, může využít komponentu *<dot:InlineScript>* a do ní napsat vlastní skript. Rámec se postará o vygenerování do stránky. [66]

DotVVM implementuje architektonický vzor „Model-View-ViewModel“ (zkráceně MVVM), který je popsán v následující podsekci. Z toho vychází také název rámce, kde „Dot“ reprezentuje rámec .NET a „VVM“ zastupuje právě návrhový vzor MVVM. [67]

Autoři DotVVM v čele s hlavním architektem Tomášem Hercegem uvádějí na svém blogu mnoho důvodů, proč se rozhodli vyvinout tento rámec – z nich je níže výběr těch nejpodstatnějších: [22]

- ASP.NET Web Forms, které byly perfektní technologií pro rychlý vývoj formulářových aplikacích, zastaraly a dostávají se do pozadí. Na nových projektech je už prakticky nelze potkat. Na trhu však chyběla stejně snadná a rychlá technologie pro vývoj formulářových aplikací a rozsáhlých systémů pro firmy.
- Pro větší projekty je využití jakékoliv technologie s klientským Javascript rámcem nepraktické. Jednak je to časově náročnější, což vede k prodlužování doby vývoje softwaru a s tím i k nárůstu ceny výsledné aplikace, a také to pro programátora znamená naučit se mnoho knihoven a podpůrných nástrojů, což může být pro méně

zkušeného vývojáře přespříliš náročné, nehledě opět na fakt, že to vede k delší době vývoje aplikace.

- Validaci musí programátor také řešit čistě ve vlastní režii – zejména v případě, kdy část validace probíhá již na klientovi a část na serveru. A v aplikacích je většinou potřeba, aby se validační chyby z klienta i ze serveru zobrazovaly stejným způsobem a uživatel nepoznal rozdíl, což vede opět k náročnějšímu vlastnímu řešení.

- Jakákoliv akce, která se jednou provede na serveru a podruhé na klientovi a její výsledek má být stejný, vede ke složitému vlastnímu řešení programátora. Příkladem může být zpracování data a času ve dvou různých případech užití. Nejdříve chce uživatel vytvořit PDF dokument, do kterého bude vloženo aktuální datum a čas. Ten je generován na serveru. Druhým případem může být zobrazení aktuálního data a času na webové stránce. Na webové stránce je aktuální datum a čas generován prostřednictvím skriptu v programovacím jazyku JavaScript. Uživatel očekává, že datum a čas v PDF dokumentu vygenerovaném na serveru bude mít stejný formát jako datum a čas na webové stránce vytvořené na klientovi pomocí programovacího jazyka JavaScript. Takové požadavku vedou většinou k náročnému vlastnímu řešení vývojáře.

Při vytváření krátkých ukázkových aplikací se u každé technologie ukázalo, že argumenty firmy Riganti, uvedené výše, mají opravdu svoje opodstatnění. Když se podíváme na všechny zdrojové kódy v předcházející části kapitoly, vidíme, že s každým přístupem lze dojít ke stejným nebo velmi podobným výsledkům. Velký rozdíl je však v náročnosti programátorovy práce. Zatímco některé přístupy vyžadují pouze znalost jazyků HTML, CSS a C# a zobrazování dat funguje pomocí provázání (anglicky „data-binding“) – například ASP.NET MVC, v jiných přístupech je nutné pracovat také s jazykem JavaScript a mnoho úsilí je věnováno pouze přebírání dat z API a jejich zobrazování v pohledu.

V DotVVM můžeme najít odkaz na ASP.NET Web Form. Technologicky je však DotVVM nesrovnatelně modernější koncept. Zatímco ASP.NET Web Forms k přenášení dat z klienta na server využívaly plné metody HTTP POST, DotVVM využívá AJAX, který probíhá asynchronně a přenáší pouze nezbytné množství informací mezi klientem a serverem. Nedochází tak k přenášení a znovunačítání celých stránek například pouze při aktualizaci dat ze serveru u klienta. [92] Další technologickou změnou je přenášení serializovaného ViewModel v JSON, a to pouze té části, která je potřeba, namísto přenášení kryptického skrytého pole ___VIEWSTATE, které bylo využíváno v ASP.NET Web Forms, jak je popsáno v sekci 2.3.1. [22]

V několika málo případech je možné narazit na komplikace, které by bylo možné označit za problémy nového rámce. Například, pokud při vytváření nového projektu ze šablony byla použita aktuálně nejnovější verze rámce .NET, což v době vzniku této práce je .NET 4.6.1., a následně se vývojář pokusí aplikaci přeložit a spustit, získá chybovou hlášku „*Could not load file or assembly 'Microsoft.Owin, Version=3.0.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The located assembly's manifest definition does not match the assembly reference. (Exception from HRESULT: 0x80131040)*“ a aplikace se nespustí. Tato chyba je způsobena pouze špatným nastavením v souboru *Web.config*, kde je uvedena starší verze knihovny OWIN, zatímco do projektu se nainstaluje novější verze. Konkrétně je ve *Web.config* uvedena verze 3.1.0.0, ale v projektu je nainstalován NuGet balíček Owin ve verzi 4.0.0.0. Tato malá chyba se dá rychle vyřešit aktualizací nastavení aplikace v souboru *Web.config*.

Co naopak je možné velmi ocenit při vytvoření nového projektu DotVVM ze šablony, je kompletně připravená architektura aplikace. DotVVM implementuje architektonický vzor MVVM popsaný v následující podsekci. Mimo přehledné architektury dle vzoru MVVM, je v novém projektu připravena i vrstva pro přístup k datům (anglicky „Data access layer“, zkráceně DAL) nebo služby (anglicky „Services“).

V ukázce níže si lze všimnout odlišností ve zdrojovém kódu pohledu oproti jiným přístupům v ASP.NET. DotVVM má vlastní syntaxi pro vkládání webových prvků. Prvek z rámce DotVVM je nutné ve zdrojovém kódu uvodit klíčovým slovem *dot* a dvojtečkou. Pro propojení „ViewModel“ a pohledu se využívá také speciální syntaxe. Pomocí složených závorek a klíčového slova *value:* lze přistoupit k vlastnosti třídy (anglicky „property“) a pomocí klíčového slova *command* lze volat metodu definovanou ve „ViewModel“. Jako typ souboru není pro pohled využíván klasický *.cshtml*, ale DotVVM používá vlastní formát *.dothtml* a *.dotmaster*, který obsahuje obsah stránky s HTML tagy obohacenou o DotVVM prvky. V ukázkovém zdrojovém kódu si lze všimnout, že programátor si vystačí se znalostí jazyků HTML, CSS a C#. Nemusí využívat řadu dalších knihoven a nástrojů, ani nepotřebuje rozsáhlý kód v jazyku JavaScript, který přenáší data z API do pohledu.

Pohled – soubor *default.dothtml*:

```
...
<dot:Content ContentPlaceHolderID="MainContent">
    <dot:Repeater DataSource="{value: Users}">
        <ItemTemplate>
            <p>{{value: _parent.Greeting}}, {{value: Name}}</p>
        </ItemTemplate>
    </dot:Repeater>
</dot:Content>
...
```

ViewModel – soubor *DefaultViewModel.cs*:

```
...
public class DefaultViewModel : MasterPageViewModel
{
    public string Greeting { get; set; } = "Ahoj";
    public List<User> Users {get; set; }

    public override Task PreRender()
    {
        var userService = new UserService();
        if (!Context.IsPostBack)
        {
            Users = userService.GetAllUsers();
        }
        return base.PreRender();
    }
}
...
```

Logika aplikace – soubor *UserService.cs*:

```
...
public class UserService
{
    public List<User> GetAllUsers()
    {
        using (var db = new MyDbContext())
        {
            return db.Users.ToList();
        }
    }
...
```

Model – soubor *Person.cs*:

```
...
public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
}
...
```

DotVVM je rámec, který určitě stojí za zvážení při tvorbě webové aplikace. Dokáže programátorovi ulehčit mechanické psaní opakujícího se kódu (pro přenos dat z API do pohledu i pro generování webových prvků) a dokáže zefektivnit komunikaci mezi klientem a serverem (AJAX s možností optimalizací, aby nedocházelo k přenášení celého „ViewModel“ v serializovaném formátu JSON). DotVVM poskytuje programátorovi kvalitní dokumentaci, která ho nejen rychle s rámcem seznámí, ale poskytne i ukázková řešení častých problémů. [67]

Architektonický vzor Model-View-ViewModel

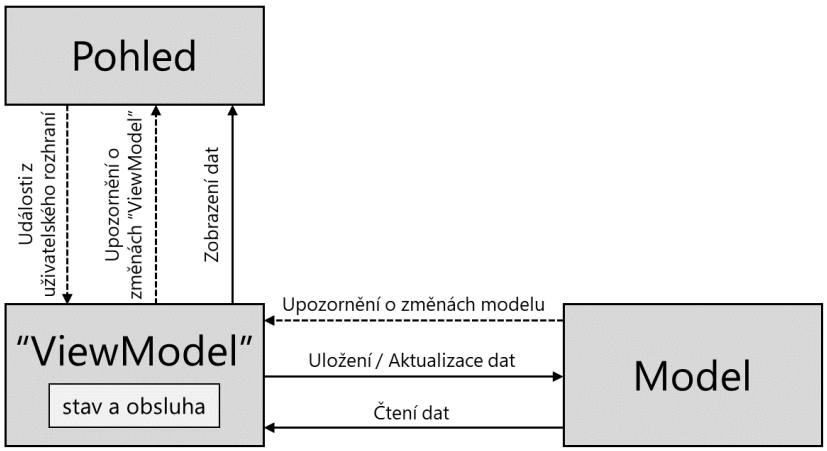
Architektura Model-View-ViewModel (MVVM), která je implementována v DotVVM, se vyznačuje rozdělením uživatelského rozhraní, stavu aplikace a dat do tří samostatných komponent: [81]

- Model („Model“) popisuje data, se kterými aplikace pracuje. Jedná se typicky o třídy, které se využívají pro vytvoření databáze přístupem „nejdříve programuj“ (anglicky „Code-First“) v rámci Entity Framework. Tento přístup znamená, že tabulky v databázi jsou vytvořeny na základě tříd z modelu aplikace.
- Pohled („View“) – stejně jako v architektonickém vzoru MVC – zajišťuje interaktivní reprezentaci dat v uživatelském rozhraní aplikace. Data získává z komponenty „ViewModel“. Pokud dojde ke změně v pohledu ze strany uživatele, informuje o tom „ViewModel“ a zpropaguje do něj změněná data.
- „ViewModel“ drží stav aplikace a jsou podle něho vykreslovány ovládací prvky aplikace do pohledu. „ViewModel“ naopak přijímá informace o změnách v pohledu a dle těchto změn se aktualizuje. Jedná se tak o nejdůležitější třídu pro běh aplikace. Součástí „ViewModel“ je kolekce *ObservableCollection<T>*, která hlásí, když je odebrán

nebo přidán její prvek, a také neméně důležité rozhraní *INotifyPropertyChanged*, které popisuje událost při změně některé vlastnosti „ViewModel“.

MVVM přináší kromě snadnější údržby a rozšiřitelnosti aplikace také zvýšení možnosti testovatelnosti aplikace.

Grafické vyjádření architektonického vzoru MVVM a komunikaci mezi jednotlivými komponentami demonstruje obrázek 2.7.



Obrázek 2.7: Architektonický vzor „Model-View-ViewModel“

Kapitola 3

Analýza požadavků na konferenční systém

Od konferenčního systému je vyžadováno uspokojení potřeb zákazníka (Zámecké návrší, p.o.). Pro uspokojení potřeb zákazníka bylo nejprve nutné zjistit a přesně vyjádřit požadavky na výslednou webovou aplikaci.

Analýza požadavků probíhala při osobních setkáních s ředitelem této organizace, který byl tou nejkompetentnější osobou pro specifikování vlastností informačního systému.

Na začátku spolupráce se zákazníkem byly zjištěny všechny aktuální požadavky. Již však při prvním setkání bylo hlavním přáním zákazníka možnost při vývoji aplikace rychle reagovat na změny v požadavcích a možnost upřesňovat specifikaci.

Kompletní požadavky jsou uvedeny níže v podkapitolách.

3.1 Funkční požadavky

Výsledná aplikace se vyznačuje svojí modularitou – je možné ji rozdělit do několika modulů, které jsou funkčně napojeny na konferenční systém jako celek. Proto i funkční požadavky jsou pro přehlednost rozděleny nejdříve na obecné požadavky a dále dle jednotlivých modulů.

Obecné požadavky

- Webová aplikace dostupná z jakéhokoliv stolního počítače, notebooku či mobilu připojeného k internetu.
- Konferenční systém bude pracovat s účty osob, a to tím způsobem, že každému uživateli bude zobrazovat relevantní informace vážící se k jeho osobě.
- V systému bude možné definovat minimálně dvě role: uživatel a správce. Správce bude mít na rozdíl od uživatele přístup do modulu „Správce“, kde bude moci zobrazit i upravit data o všech uživatelích. Podrobněji popsáno dále v jednotlivých sekcích.

Registrace na konferenci a správa uživatelů

- Uživatel provádí registraci sám. Při registraci je nutné vyplnit formulář, který obsahuje povinné i nepovinné položky.

Mezi povinné údaje patří: jméno, příjmení, email, telefonní číslo, pracovní pozice a udělení souhlasu se zpracováním osobních údajů a s pořizováním audiovizuálního záznamu se svojí osobou za účelem propagace akce.

Mezi nepovinné údaje patří: titul před jménem, titul za jménem, datum narození, místo narození, poznámka, údaje o organizaci (IČ, DIČ, Název organizace, Ulice a číslo popisné, Obec, PSČ) a požadavek zaslání potvrzení o platbě.

- Registrační stránka nesmí být chráněna heslem (resp. přístupovými údaji) a musí být dostupná komukoliv, kdo zná URL nebo je přesměrován z jiné webové stránky.
- Po registraci je uživateli zaslán email s potvrzením úspěšné registrace obsahující i informace potřebné k provedení platby účastnického poplatku.
- Konferenční systém umožňuje přihlášení a následné odhlášení účastníka, který provedl registraci. Jako přihlašovací údaje slouží jeho email a heslo zadané při registraci.
- V případě zapomenutí přihlašovacích údajů je možné přes konferenční systém zažádat o jejich zaslání na email. Tento proces je plně automatický.
- Správce má právo zobrazit i upravit data o uživatelích a také uživatele odstranit. Dále je nutné mít možnost exportovat data do souboru ve formátu xlsx.
- Zobrazení dat o uživatelích bude možné v různých náhledech, a to konkrétně: „Přehled“, „Fakturační údaje“ a „Všechny sloupce“. Každý náhled bude tematicky zaměřen a bude vybírat pouze data relevantní k tématu daného zobrazení (např. Zobrazení „Fakturační údaje“ bude zobrazovat právě ty údaje potřebné pro vystavení faktury).
- Správce bude mít možnost zobrazit celkový počet zaregistrovaných osob.

Registrace na jednotlivé semináře a správa registrací

- Konference se skládá z přednášek a seminářů. Přednášky jsou dostupné všem účastníkům, semináře jsou limitovány kapacitou místností.
- U semináře je třeba uchovávat tyto informace: jméno semináře, datum, čas, jméno přednášejícího, místnost a kapacita místnosti.
- Semináře jsou rozděleny do bloků. V jednom časovém bloku může probíhat několik seminářů.
- Účastník má možnost se zaregistrovat pouze na jediný seminář v konkrétním časovém bloku.
- Účastník může svoji registraci na seminář upravit či zrušit kdykoliv do data ukončení registrací.
- Registrace na daný seminář je povolena pouze pokud nebyla ještě naplněna jeho kapacita (myšlena kapacita místnosti).
- Správce musí mít možnost zobrazit obsazenost seminářů, a to ve dvou různých náhledech na data: zobrazení vybraných seminářů k účastníkovi a zobrazení účastníků k vybranému semináři. U druhého zobrazení je také požadován celkový počet uživatelů, kteří jsou zaregistrováni na daný seminář a informace o naplnění kapacity.

- Je třeba mít možnost exportovat data o seminářích a jejich obsazenosti do souboru ve formátu xlsx.

Volba stravování

- Účastníci i správci budou mít možnost zvolit si stravování na konferenci.
- Stravování se dělí do kategorií (např. neděle – večeře, neděle – ochutnávka vína apod.). Kategorie stravování může mít několik voleb (např. menu 1, menu 2).
- Účastníci i správci musí mít možnost označit zájem o danou kategorii stravování a pokud kategorie obsahuje také volby, musí vybrat příslušnou volbu stravování z dané kategorie.
- Uživatelé musí mít možnost upravovat i smazat svůj výběr stravování až do dne, kdy bude volba stravování ukončena.
- Správce má možnost zobrazit vybrané možnosti stravování, a to ve dvou různých náhledech na data: zobrazení vybraného stravování k účastníkovi a zobrazení účastníků k vybrané kategorii (a volbě) stravování. U druhého zobrazení je také požadován celkový počet uživatelů, kteří si vybrali danou volbu stravování.
- Je třeba mít možnost exportovat data o stravování do souboru ve formátu xlsx.

3.2 Nefunkční požadavky

Nefunkční požadavky se většinou dotýkají informačního systému jako celku nebo všech modulů. Proto již nejsou dále členěny, ale jsou představeny v následujícím seznamu.

- Responzivní design – rozložení zobrazovaných prvků a jejich vzhled se musí přizpůsobit zařízení, na kterém je aplikace zobrazována. Minimální uvažovaná úhlopříčka displeje takového zařízení je 5", což odpovídá 12,7 cm.
- Konferenční systém bude implementovat postupy potřebné pro zabezpečení dat i celé aplikace, a to zejména autentizace, autorizace a šifrování (minimálně hesel).
- Doba odezvy systému nepřekročí hranici 2 sekundy pro 95% žádostí. Ve zbylých 5% případů nepřekročí průměrná doba odezvy 10 sekund.
- Systém bude navržen jako dlouhodobě udržitelný. Za dlouhodobě udržitelný lze považovat systém v případě, kdy bude možné implementovat v budoucnu další modul do konferenčního systému bez zásahu do již dokončených modulů. Stejně tak je nutné, aby bylo možné modul, který nesouvisí se správou uživatelů, kdykoliv odstranit či deaktivovat bez většího zásahu do celého systému.
- Konferenční systém musí být dostupný minimálně 95% celkového času. Celkový čas je brán jako časový interval od spuštění registrací na konferenci do uplynutí 2 měsíců od ukončení konference.

3.3 Bezpečnost aplikace

V konferenční systému vyvíjeného pro Zámecké návrší, p.o. se předpokládá práce s osobními údaji, a to nejen osob, které se na akci hlásí jako účastníci, ale také osob, které vystupují v systému jako správci. Zároveň se také předpokládá reálné nasazení webové aplikace. Z těchto důvodů je nutné, aby tento informační systém poskytoval dostatečné zabezpečení osobních údajů a splňoval normy GDPR.

Základní bezpečnostní požadavek je zaměřen na autentizaci a autorizaci. Pro vstup do informačního systému se uživatel musí autentizovat. Jedinou výjimku tvoří registrační formulář, který autentizaci ani autorizaci nevyžaduje.

Aplikace je dále rozdělena na sekci pro účastníky a sekci pro organizátory. Na základě autorizace mají uživatelé i organizátoři přístup pouze do své sekce.

Komunikace s webovou aplikací i uložená data by měly být šifrované z důvodu zajištění ochrany uživatele a jeho osobních údajů.

Pro zajištění bezpečnosti osobních dat při vývoji systému je nutné rozdělení vývojového a produkčního prostředí. Zejména poté využití produkční a testovací databáze. Zatímco produkční databáze obsahuje chráněné osobní údaje, v testovací verzi jsou využívány údaje fiktivních osob.

Stav aplikace (aktuální webové stránky) je uchováván ve „ViewModel“, ze kterého může probíhat přímá komunikace s databází. Z hlediska bezpečnosti není přímá komunikace s databází z „ViewModel“ optimální řešení. Jelikož data z databáze načtená do „ViewModel“ si může útočník zobrazit ve vývojářských nástrojích prohlížeče, mohou tak uniknou například osobní údaje, či jiná data, která nemusí být na dané webové stránce ani zobrazena. Z tohoto bezpečnostního důvodu je do aplikace přidána ještě mezivrstva nazývaná „Data Transfer Objects“, zkráceně DTOs, která svou strukturou odpovídá struktuře entit v databázi (mají stejné vlastnosti). Pomocí nějaké metody ve službách (anglicky „Service“, což je třída obsahující metody nejčastěji pro přenos dat mezi DTOs a databází, ale slouží také pro jakékoliv zpracování dat z DTOs) jsou data načtena ze záznamu v databázi do DTO, přičemž jsou načtena jen ta data, která budou skutečně potřeba (například nemusí být vůbec načten otisk (anglicky „hash“) hesla) a do „ViewModel“ aktuální stránky se zašle pouze DTO obsahující jen potřebné a bezpečné informace. Díky tomu „ViewModel“ nedostává nikdy data přímo z databáze.

3.4 GDPR

General Data Protection Regulation (zkráceně „GDPR“, v českém překladu „Obecné nařízení o ochraně osobních údajů“) je legislativa Evropské unie, která upravuje pravidla pro zpracovávání osobních údajů. GDPR bylo schváleno 27. 4. 2016 a v platnost vstoupilo 25. 5. 2018. [18]

Přestože GDPR není legislativa stanovující pravidla pouze pro informační systémy, ve kterých je možné nakládat s osobními údaji, velmi úzce se oblasti informačních technologií dotýká, jelikož v dnešní době již většina osobních údajů je zpracovávána elektronicky.

V době vzniku této bakalářské práce je GDPR novou problematikou, která je pro mnoho firem a institucí stále neznámá a na kterou nejsou připravené. [33] [84] [35] GDPR je v firemní sféře velký tématem i proto, že pokuty za nesplnění požadavků této legislativy šplhají až k 20 000 000 € nebo k 4% z celkového ročního obratu společnosti. [18]

Kapitola 4

Návrh informačního systému

Návrh informačního systému má za cíl přenést požadavky a specifikaci aplikace od zákazníka do takového popisu, na základě něhož lze vytvořit výslednou aplikaci. Výstupem této fáze vývoje softwaru jsou diagramy (např. ER diagram) a technická specifikace.

Ian Sommerville ve své knize Softwarové inženýrství definuje návrh softwaru jako kreativní aktivitu, při které se na základě požadavků zákazníka identifikují softwarové komponenty a jejich vztahy. [74]

Součástí návrhu informačního systému může být také výběr implementační platformy a konkrétních technologií. [9]

V následujících částech kapitoly je popsán výběr implementačních technologií, návrh databáze a návrh aplikace. Kromě popsání vybraných technologií je v kapitole cílem také pečlivě zdůvodnit zvolený výběr technologií.

4.1 Výběr implementačních technologií

Jedním z hlavních požadavků na informační systém bylo, aby aplikace byla vytvořena jako webová. Na platformě .NET je pro webový vývoj určen rámec ASP.NET.

Prvním otázkou při základním výběru implementačních technologií může být, proč zvolit platformu .NET a ne například některý z řady rámců pro vývoj webových stránek v PHP (Laravel, Symfony2, Nette, CodeIgniter... [73]) či Python (Django, Flask, Tornado, Falcon... [61]).

Na tuto otázku snad ani neexistuje odpověď. Už jenom proto, že se nedá říci, která technologie je lepší a která horší. Vždy záleží na konkrétních požadavcích na aplikaci a také na programátorovi, který rámec či technologie mu vyhovují více.

Obecně však nastíním několik důvodů, proč v tomto případě dávám přednost platformě .NET (respektive ASP.NET) před konkurenčními technologiemi:

- Za .NET platformou stojí silná společnost Microsoft, která u svých produktů zajišťuje dlouhodobou podporu a rozsáhlou technickou dokumentaci. Kromě silné společnosti stojí za .NET velká skupina vývojářů, kteří jsou aktivní na fórech a diskuzích. Obecně se jedná o technologii, ke které je snad vždy možné dohledat nápovědu a podporu. [48]

- Pro vývoj nejen na platformě .NET lze využít Visual Studio – pokročilé vývojové prostředí od společnosti Microsoft. [50] Toto vývojové prostředí poskytuje vývojáři téměř veškeré nástroje a funkcionality k efektivnímu vývoji aplikace. Výborným doplňkem, který tuto funkcionalitu ještě rozšíří, je ReSharper. [29]

- Visual Studio nabízí přenesení výsledné aplikace do Microsoft Azure (cloud od společnosti Microsoft). Tím, že obojí vyvíjí jedna společnost, je provázanost vývoje v .NET a cloudu Azure na vysoké úrovni. V Microsoft Azure lze jednoduše hostovat aplikace, škálovat je a nastavovat jim optimální hostovací prostředí. [47]

- Platforma .NET nabízí podporu pro dlouhodobou udržitelnost aplikace. Stále vycházejí nové verze, které vylepšují ty předchozí. Je možné do projektu doinstalovávat nové funkcionality v podobě NuGet balíčkům či stávající balíčky aktualizovat. Díky tomu je možné funkčnost aplikace rozšiřovat a modernizovat. [49]

- Nejčastějším implementačním jazykem aplikací na platformě .NET je C#. Jedná se o moderní objektově orientovaný jazyk, který nabízí vývojáři možnost tvorby přehledného a dobře čitelného kódu. C# se také stále modernizuje a přináší nové konstrukce jazyka a novou funkčnost. Aktuálně nejnovější verzí je C# 7.2. [85]

Nelze říci, že PHP či Python rámce nesplňují některé z bodů výše a rozhodně nelze říci, že v PHP by nešla vytvořit stejně kvalitní výsledná aplikace. Avšak na základě osobních zkušeností s .NET i s PHP a Python rámci dávám nyní přednost při větším projektu platformě .NET (a programovacímu jazyku C#) a při menším projektu rámci CodeIgniter (a programovacímu jazyku PHP). Jelikož informační systém pro Zámecké návrší dle požadavků patří do většího a dlouhodobě udržovaného systému, vybral jsem platformu .NET (respektive její webový rámec ASP.NET).

Samotný webový rámec ASP.NET nabízí několik programátorských přístupů či rámců k vytvoření webové aplikace. Jednotlivé přístupy byly představeny v sekci 2.3 – Webový vývoj v .NET.

Na základě informací o jednotlivých přístupech bylo cílem vybrat nejvhodnější implementační technologii. Informace a hodnocení jednotlivých přístupů ve výběru níže se opírá o informace uvedené v sekci 2.3 – Webový vývoj v .NET. Informace v této sekci jsou podloženy zdroji uvedenými v části Literatura. Níže jsou stručně okomentovány všechny představené přístupy s cílem zhodnotit jejich vhodnost pro požadovaný informační systém.

ASP.NET Web Forms by nebyly vhodnou technologií z důvodu jejich zastaralosti. Vývoj by byl sice velmi rychlý, ale princip fungování tohoto rámce již nevyhovuje moderní aplikaci, která cílí na výkon, možnost automatického testování a responzivitu.

ASP.NET Single Page Application je velmi zajímavý přístup. Jistě by uživatelé ocenili, když by přechod mezi jednotlivými stránkami aplikace byl okamžitý. Vzhledem k tomu, že se očekává práce s velkým množstvím dat a pohledů a že aplikace se dle specifikace jeví jako rozsáhlý informační systém, přístup ASP.NET SPA by nebyl vhodný z důvodu velké zátěže RAM paměti klienta.

ASP.NET Web API v kombinaci s „front-end“ knihovnami by nepřijatelně prodloužilo dobu vývoje výsledné aplikace z důvodu nutnosti vytvořit veškeré pohledy čistě v režii vývojáře bez podpory rámce a také nutnost stále programovat přenos dat mezi API a pohledem, což by také bylo zbytečně pracné, jelikož se v aplikaci předpokládá zejména zpracování dat a tudíž komunikace s API a přenos velkého množství dat by se vyskytoval téměř u každého pohledu.

ASP.NET Web Pages jsou jen těžko překonatelný nástroj – jak již název napovídá – k vytvoření jednoduchých webových aplikací. Doporučit je pro vývoj informačního systému však nelze. Tento přístup neimplementuje žádný pokročilý architektonický vzor. Při tvorbě velké aplikace v ASP.NET Web Pages by tak brzy mohlo docházet k nepřehlednosti kódu a výrazně by se zhoršila i udržitelnost aplikace.

Na výběr a porovnání zůstávají rámce ASP.NET MVC a DotVVM. První odlišnost najdeme v architektuře, kterou implementují. Zatímco ASP.NET MVC implementuje architekturu MVC, DotVVM implementuje MVVM. Architektonický vzor MVVM nabízí jednodušší a přímější provázání vzhledu a logiky díky přímým datovým propojením (anglicky „data-binding“) mezi „View“ a „ViewModel“. Je tak možné snadněji manipulovat s daty v pohledu než v architektonickém vzoru MVC, kde manipulace z daty jde přes řadič. [76]

Další rozdíl najdeme v zaměření rámců. Zatímco ASP.NET MVC se zaměřuje obecně na jakékoliv webové aplikace, DotVVM se specializuje na rozsáhlé aplikace typu informačních systémů.

Jako vhodnou implementační technologii jsem vybral ASP.NET s rámcem DotVMM.

Výsledek mého výběru podpořil i fakt, že DotVVM je relativně nový rámec – první verze byla představena v roce 2015 – a mým cílem bylo rámec důkladně poznat a zhodnotit jeho konkurenceschopnost oproti jiným rámcům založených na ASP.NET.

Vzhledem k vybranému rámci DotVVM jsem jako implementační jazyk zvolil C# pro „back-end“ a HTML, CSS a minimálně také JavaScript pro „front-end“. Výběr byl jednoduchý, jelikož rámec podporuje právě tyto programovací a značkovací jazyky.

Vzhledem k implementačnímu jazyku C# a vývoji v ASP.NET bylo jako vývojové prostředí využito Visual Studio, do kterého byl doinstalován doplněk pro DotVVM. Díky tomuto rozšíření bylo například možné vytvořit nový projekt přímo jako DotVVM projekt s již připravenou prázdnou šablonou.

Jako hosting se nabízel Microsoft Azure, který byl nakonec i využit. Nasazení aplikace do Azure je díky přímé provázanosti Visual Studia s Microsoft Azure snadné a přívětivé pro vývojáře. Microsoft Azure nabízí také dostatečný výkon a garanci dostupnosti služeb na dostatečně vysoké úrovni i pro náročná komerční řešení.

Obecně lze říci, že po zvolení rámce DotVVM pro vývoj nebylo těžké vybrat další náležitosti jako implementační jazyk, vývojové prostředí a hosting. ASP.NET, C#, Visual Studio a Microsoft Azure jsou vzájemně tak provázány, že při využití jednoho z vyjmenovaných ve vývoji je výhodné a snadné využít i další a použít tak kompletní řešení na platformě společnosti Microsoft.

4.2 Návrh databáze

Databáze slouží k perzistentnímu, organizovanému, dlouhodobému a bezpečnému uložení strukturovaných dat. V informačním systému pro Zámecké návrší, p.o. existuje několik základních entit, které je potřeba dlouhodobě uchovávat v databázi:

- **Uživatel** je základní entita v databázi. Kromě údajů o uživateli je nutné uchovávat také informace související s jeho účtem v aplikaci:
  - role (účastník konference, pořadatel konference nebo administrátor)
  - tvrzení o uživateli neboli vlastnosti (anglicky „claim“)
  - přihlašování
- **Organizace**, ke které účastník konference patří. Z jedné organizace se může konference zúčastnit více účastníků.
- **Jídlo**, které si účastník konference objednal. Jídlo je rozděleno do kategorií. Z každé kategorie si účastník může vybrat buď právě jednu položku nebo více položek. Záleží na nastavení dané kategorie. Například při výběru polévky si může vybrat pouze jednu

z nabízených, ale při výběru moučníku si může vybrat i několik. V obou případech si účastník nemusí vybrat žádnou z nabízených možností.

- **Workshopy a přednášky**, na které se účastník přihlásil. Workshopy a přednášky jsou rozděleny do bloků dle času. V jednom časovém bloku si účastník konference může vybrat maximálně jednu přednášku či workshop.

Výsledné schéma databáze založené na požadavcích a popisu výše je zobrazeno na obrázku 4.1.

Pro správu uživatelů (jejich registraci, přihlašování, odhlašování, reset hesla) je využíván systém členství ASP.NET Identity. Jména databázových tabulek, které potřebuje tento systém členství pro správnou funkčnost, začínají textem *AspNet*. Díky tomu je možné je v návrhu databáze dobře rozlišit.

Obrázek 4.1: Schéma databáze

4.3 Návrh architektury aplikace

Návrh architektury představuje důležitou fázi vývoje aplikace zejména v případě potřeby vývoje dlouhodobě udržitelné aplikace. V případě dobrého architektonického návrhu je možné výslednou aplikaci jednoduše rozšířit například na mobilní platformu s využitím naprogramovaných služeb z existující aplikace. Nemusí se však jednak pouze o rozšíření aplikace na jinou platformu. Při kvalitním architektonickém návrhu je i přidání malé funkcionality dobře proveditelné, jelikož je zřejmé, do jaké části zdrojového kódu je třeba požadovanou funkcionalitu doprogramovat.

V následující části jsou představené jednotlivé vrstvy aplikace a komunikace mezi nimi: [42]

- **Datová vrstva (anglicky „Data access layer“, zkráceně DAL)** popisuje model. Uchovává v sobě třídy popisující entity aplikace, nastavuje vytváření tabulek databáze z objektů v aplikaci (tzv. „Code-First“ přístup při vytváření databáze – podrobně popsáno v implementaci databáze) a také obsahuje metodu pro inicializaci databáze – pro vložení počátečních dat. Poskytuje prostředky pro práci s daty v databázi. Pokud je třeba přidat vlastnost entitě nebo ji odebrat, změnit název připojovacího řetězce (anglicky „connection string“) do databáze nebo změnit inicializaci databáze, je to možné provést ve vrstvě DAL.

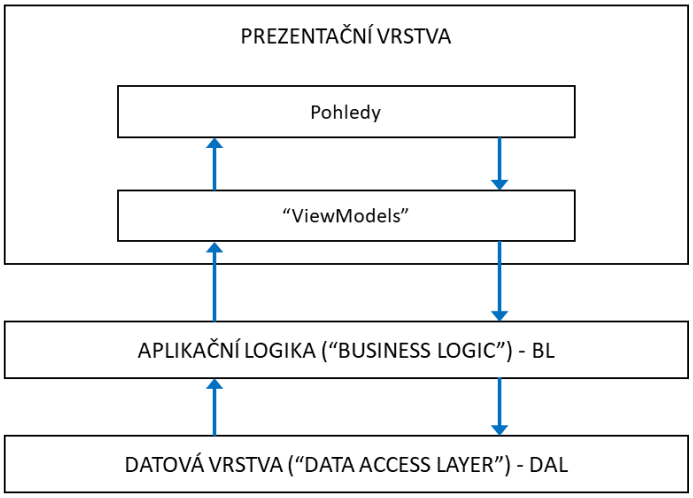
- **Aplikační logika (anglicky „Business logic“, zkráceně BL)** provádí operace nad daty v informačním systému. Ukládá v sobě definici „Data Transfer Objects“, které jsou využívány pro přenos dat z databáze do „ViewModel“ – více v sekci 3.3. BL dále obsahuje služby (anglicky „Services“), což jsou třídy poskytující typicky zpracování dat nebo mapování dat mezi entitami z databáze a DTOs. Služby jsou volány při potřebě načtení, uložení nebo smazání dat z databáze. Můžou však sloužit i pro zpracování dat, při kterém nedochází ke komunikaci s databází. Například zpracování časových dat.

- **Prezentační vrstva (anglicky „Presentation Layer“)** je vrstva, která zajišťuje udržení stavu aplikace a zobrazení dat a zpracování uživatelských akcí. Lze ji z hlediska návrhu architektury rozdělit ještě na 2 vrstvy:

  - **„ViewModels“** uchovávají stav aplikace, načítají data pro zobrazení a obsahují obslužné metody pro uživatelské akce. Dále volají potřebné služby – ať už pro načtení dat nebo naopak jejich uložení, či pouze zpracování.
  - **Pohledy (anglicky „Views“)** definují interaktivní uživatelské rozhraní aplikace. Zobrazují statická data, data z „ViewModels“ a ovládací kontrolky. Do této vrstvy lze zařadit nejen samotný soubor ve formátu *.dothtml*, ale i stylopisy, fonty, obrázky, klientské skripty a další program související se zobrazením dat a uživatelskou interakcí.

Další část aplikace, která se však již neřadí přímo do návrhu architektury, tvoří **testy**. Ty jsou důležitou částí každé moderní dlouhodobě udržovatelné aplikace. Testy zajišťují ověření funkčnosti aplikace. Při změně některé funkcionality informují vývojáře, zda nedošlo ke zrušení jiné funkčnosti (například v nějakém krajním případu). Zároveň díky nim lze při vývoji aplikovat testy řízený vývoj (anglicky „Test-Driven Development“)

Architektonický návrh aplikace s vyznačením jednotlivých vrstev je zobrazen na obrázku 4.2. Šipky znázorňují směr komunikace mezi jednotlivými vrstvami.



Obrázek 4.2: Návrh architektury aplikace

Kapitola 5

Implementace a testování

Kapitola se zabývá popisem konkrétních programátorských řešení, technologií, knihoven a NuGet balíčků využitých při vývoji aplikace. Zaměřuje se na zajímavé přístupy, které vedou k zajištění časově efektivního vývoje a zvýšení dlouhodobé udržitelnosti, provozuschopnosti a bezpečnosti aplikace.

Celý konferenční informační systém je vytvořen jako jedno řešení ve vývojovém prostředí Microsoft Visual Studio Enterprise 2017 ve verzi 15.6.3. Jednotlivé vrstvy programu – představené na obrázku 4.2 – jsou implementovány jako samostatné projekty mezi nimiž jsou vytvořeny reference. Všechny projekty jako cílovou verzi platformy .NET využívají verzi „**.NET Framework 4.5.2**“. Jako spouštěcí je nastaven projekt *ConferenceSystemApp*, který odpovídá vrstvě „Prezentační vrstva“ v návrhu architektury.

V průběhu vývoje je informační systém spouštěn a testován na serveru IIS Express na lokálním počítači. V produkčním prostředí je pro hostování aplikace využíván „App Service“ v Microsoft Azure.

Následující kapitoly představují nejdříve implementační podrobnosti jednotlivých vrstev aplikace. Následně je představen testy řízený vývoj (anglicky „Test-Driven Development“, zkráceně TDD) a jeho přínosy i nevýhody pro vývoj aplikace. TDD je představováno na základě využití tohoto přístupu v programování části funkcionality informačního systému.

5.1 Datová vrstva aplikace a databáze

Datová vrstva slouží pro definici objektů využívaných v aplikaci a k definici entit v databázi, které těmto objektům odpovídají. Zároveň poskytuje přístup k datům uloženým v databázi bez nutnosti vytváření SQL dotazů. K tabulkám a záznamům z databáze je možné přistupovat jako k objektům. Místo psaní SQL dotazů (*INSERT, DELETE, UPDATE*) je možné s tabulkami v databázi pracovat v programu skrze třídy, které dané entity reprezentují a jsou s nimi provázané.

Pro propojení objektů s entitami a pro práci s databází je využíván Entity Framework (EF), což je technologie od společnosti Microsoft zajišťující objektově-relační mapování. [46] Díky EF je možné objekty z aplikace přenést do relačního schématu databáze. Do řešení je EF přidán jako NuGet balíček *EntityFramework* ve verzi 6.1.3. Pro propojení objektů v projektu se schématem databáze je využit přístup *Code First*, což je přístup, kdy jsou nejdříve vytvořeny objekty v aplikaci a poté jsou přeneseny do relační databáze jako entity po zadání příkazů *Enable-Migrations*, *Add-Migration "Nazev_migrace"* a *Update-Database* do *Package Manager Console*.

Pro testovací účely je využívána MSSQL databáze přístupná na lokálním SQL Serveru, který je ve verzi 13.0.4001. V produkční verzi aplikace je připojena SQL databáze dostupná na SQL serveru hostovaném v Microsoft Azure. [47] Ve zdrojovém kódu aplikace je nastavený připojovací řetězec (anglicky „Connection String“) na lokální databázi. Z důvodu bezpečnosti není připojovací řetězec na produkční databázi v kódu uveden, ale nastavuje se až v hostovacím prostředí Microsoft Azure po publikování projektu.

Pro správu uživatelů (jejich registraci, přihlašování, odhlašování, reset hesla) je využíván systém členství ASP.NET Identity přidaný do projektu jako NuGet balíček *Microsoft.AspNet.Identity.EntityFramework* ve verzi 2.2.1. Tento systém členství vyžaduje pro správnou funkčnost konkrétní databázové tabulky pro ukládání informací o uživatelích. [21] Názvy těchto tabulek začínají textem *AspNet*. Díky tomu je možné je v databázi dobře rozlišit.

5.2 Aplikační logika

Aplikační logika rozhoduje, jakým způsobem mohou být data vytvořena, uložena nebo změněna. K provádějí operací nad daty v informačním systému slouží služby (anglicky "Services"), což jsou třídy s metodami zpracovávající data.

Služby jako jediné mají možnost ukládat nebo číst data z databáze. Pokud webová stránka potřebuje načíst data z databáze, musí její „ViewModel“ zavolat některou z metod dané služby, aby jí data poskytla. Stejný postup platí i pro ukládání. Služby zajišťují, že při ukládání dat budou dodržena veškerá pravidla pro zachování integrity databáze.

Každá služba nese jednu odpovědnost. Například služba „*WorkshopService*“ obsahuje metody potřebné právě a jenom pro obsloužení registrace workshopů v informačním systému.

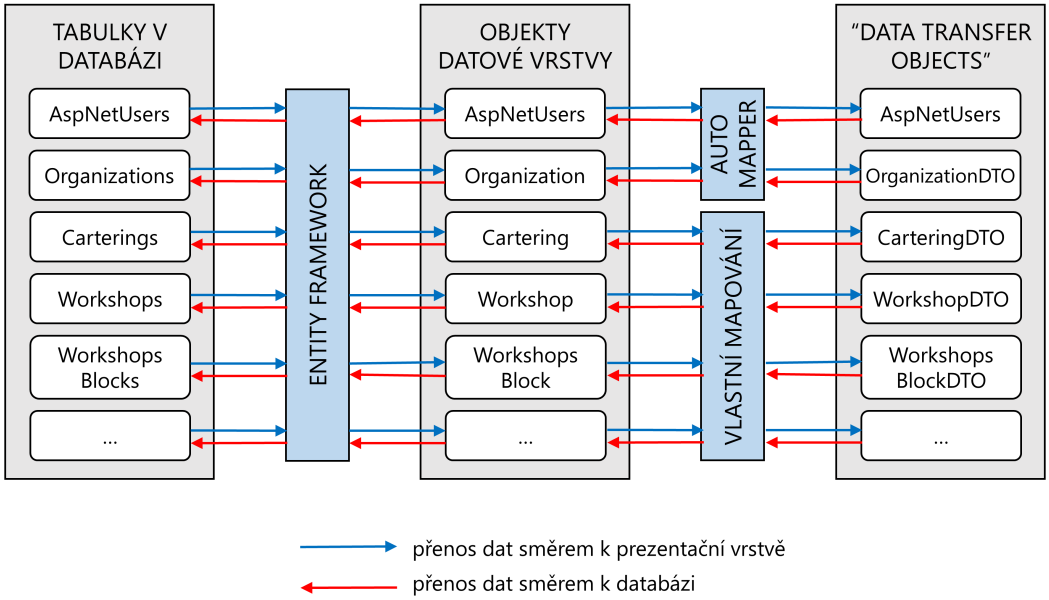
Kromě služeb obsahuje vrstva aplikační logiky také „Data transfer Objects“, zkráceně DTOs, což jsou objekty sloužící pro transformaci dat mezi jednotlivými vrstvami aplikace. Konkrétně jsou do DTOs načítána potřebná data z objektů na datové vrstvě a po načtení jsou DTOs předány prezentační vrstvě, aby z nich mohla přečíst data potřebná k zobrazení uživateli. Tyto transformační objekty, které slouží pouze k přesunu dat mezi vrstvami, jsou implementovány z důvodu bezpečnosti aplikace. Nikdy se tak v aplikaci nestane, že by byl objekt z datové vrstvy – nesoucí kompletní data ze záznamu z databáze – předán do prezenční vrstvy, kde by z něj mohl útočník přečíst například otisk (anglicky „hash“) hesla.

Každá třída z datové vrstvy má odpovídající třídu ve vrstvě aplikační logiky se stejným jménem a příponou *DTO*, která má stejná jména vlastností se stejnými datovými typy. Například třída *AppUser* má svoji transformační třídu *AppUserDTO*.

Mapování mezi objekty datové vrstvy a DTOs v režii programátora by vedlo k mechanickému a zdlouhavému psaní kódu, ve kterém je možné snadno udělat chybu a tím potenciálně snížit bezpečnost aplikace. Pro mapování mezi objekty je v konferenčním systému využívána knihovna AutoMapper ve verzi 6.2.2 doinstalovaná jako NuGet balíček. Pro správnou funkčnost je při startu aplikace volána metoda *Initialize()* v statické třídě *MapperInitializer*. Ta nastaví, které objekty se mapují na které a jaká pravidla při mapování platí (například jaké vlastnosti objektu se při mapování ignorují). Pokud se ve dvou mapovaných objektech jmenují vlastnosti stejně, poté AutoMapper zařídí správně předání hodnot sám. Pokud jsou však jména vlastností různá, je potřeba při inicializaci nastavit mapování manuálně a definovat, které vlastnosti různých objektů si navzájem odpovídají. Při potřeba mapovat objekty v některé službě již stačí se odkázat na mapu vytvořenou

při inicializaci (při startu aplikace). Mapa je název pro skupinu pravidel určujících, jaké objekty se mapují na jaké a jakým způsobem. [75]

Způsob přenosu dat z databáze až do DTOs, ze kterých čte prezentační vrstva aplikace, a obráceně při zápisu dat z prezentační do databáze je znázorněn na obrázku 5.1. Šipky naznačují směry předávání dat.



Obrázek 5.1: Způsob přenosu dat z databáze do DTOs, ze kterých čte prezentační vrstva, a obráceně při zápisu dat z prezentační vrstvy do databáze

Kromě transformačních tříd pro třídy datové vrstvy je možné jako DTOs vytvořit třídy, které nemají odpovídající třídu v datové vrstvě, ale které slouží pro transformaci dat do jiné struktury. Tento přístup je v konferenčním systému využit pro zobrazení souhrnných dat o počtu zaregistrovaných lidí na workshopy a počty přihlášených porcí jídel. Databáze zmíněné informace obsahuje nikoliv však v podobě vhodné pro statistické zobrazení. Proto jsou vytvořeny speciální DTOs, které svou strukturou jsou připraveny nést informace o názvu kategorie (například daného workshopu) a počtu lidí, který je k tomuto workshopu přihlášený. Počet lidí se získává pomocí sumarizačních dotazů nad načtenou kolekcí objektů.

Specifický problém bylo nutné vyřešit při registraci workshopů. Každý workshop má omezenou kapacitu a registrace na ně se spouští pro všechny uživatele ve stejnou chvíli. Při zaregistrování uživatele na workshop dojde u tohoto workshopu ke snížení volné kapacity o 1. Jakmile je dosaženo nulové volné kapacity, již není možné se na tento workshop registrovat. Zbývající volná kapacita je uložena v databázi u každého záznamu jako atribut tabulky *Workshops*.

Představený postup by mohl vést k porušení pravidel registrace a k umožnění registrace většího počtu účastníků na jeden workshop, než kolik pravidla dovolují. Je to způsobeno tím, že když by byly současně obsluhovány dva požadavky, tak oba si načtou aktuální hodnotu volné kapacity (např. 1), u obou uživatelů se provede registrace (propojení v databázi mezi daným uživatelem a daným workshopem), sníží se u obou kapacita volných míst na 0 a tato nová hodnota se dvakrát zapíše do volné kapacity. Došlo by tak ke snížení volné kapacity z 1 na 0, ale současné registraci dvou různých uživatelů, což nelze připustit.

Tuto problematiku lze obecně řešit pomocí zamykání zdroje. [86] V konferenčním systému je ve službě *WorkshopService* vytvořen statický objekt *RegisterWorkshopLock* určený pouze ke čtení. Při zavolání metody *RegisterWorkshop* je tento objekt uzamčen a je uvolněn až po doběhnutí dané metody. Pokud by ve chvíli, kdy je objekt *RegisterWorkshopLock* uzamčen, někdo další zavolal metodu *RegisterWorkshop*, bude muset vyčkat na uvolnění zámku. Díky tomuto řešení není umožněna více než jedna registrace na workshop v konkrétní čas. Jedna registrace trvá dostatečně krátkou dobu na to, aby uživatel nepoznal zpoždění ani v případě, že by musel čekat na uvolnění zámku.

Do aplikační logiky patří i emailové služby, které zajišťují automatické rozeslání emailů po registraci. Tento požadavek, který vede na výraznou automatizaci procesů souvisejících s organizací konference lze implementovat několika různými způsoby. První možností je vytvoření samostatné služby v Microsoft Azure, která budou předány přes její API všechny potřebné údaje a tato služba zajistí rozeslání elektronické pošty. Tato služba se jmenuje „SendGrid Email Service“ a přes ní je možné za měsíc zdarma rozeslat až 25 000 emailů. [77] V konferenčním systému byla zvolena přímější možnost, kdy je přímo v aplikaci vytvořen SMTP klient za využití třídy *System.Net.Mail*, kterému je při volání jeho konstruktoru předáno doménové jméno emailového serveru a port, na kterém jsou dostupný SMTP server. Při inicializaci SMTP klienta je třeba ještě nastavit přihlašovací údaje a v konferenčním systému dojde v tomto kroku také k povolení SSL protokolu při komunikaci – opět z důvodu bezpečnosti a důvěryhodnosti emailové komunikace. Dále je vytvořena e-mailová zpráva jako nová instance třídy *System.Net.Mail.MailMessage*, které je v konstruktoru nastaveno tělo, předmět, příjemce a případně další vlastnosti e-mailové zprávy. Následným voláním metody *Send* dojde k odeslání emailové zprávy příjemci. [45]

Poslední částí aplikační logiky je správa uživatelů s pomocí systému členství ASP.NET Identity. [21] Třída *AppUserManager* poskytuje metody pro registraci uživatele, přihlášení uživatele, přiřazení role a další. V této knihovně je řešeno například hašování hesel a uložení otisku hesla místo samotného hesla, což je důležitý aspekt bezpečnosti celého informačního systému. Pro hašování hesla je využívána metoda *HashPassword* ze třídy *Microsoft.AspNet.Identity.PasswordHasher*.

5.3 Prezentační vrstva

Prezenční vrstva zajišťuje interakci s uživatelem. Nejenže zobrazuje formátovaná data, ale dává uživateli také možnost akcí v systému, tyto akce zpracovává a vyvolává na ně adekvátní reakce. [44]

Prezentační vrstva se dělí dále do pohledů a „ViewModels“. Při použití rámce DotVVM jsou pohledy ve zdrojovém kódu soubory s koncovkou *.dothtml*, případně *.dotmaster*, což jsou HTML soubory s možností obohacení zdrojového kódu o DotVVM webové prvky. Každá webová stránka má svůj pohled a „ViewModel“. Pohled definuje uživatelské rozhraní a „ViewModel“ definuje obsluhu webových kontrolek, volá služby pro získání nebo naopak uložení dat, zpracovává a reaguje na uživatelský vstup. Kromě pohledů a „ViewModels“ je v prezentační vrstvě uložen další obsah vztahující se k uživatelskému rozhraní. Konkrétně pro konferenční systém se jedná o obrázky, fonty, stylopisy a doplňkové skripty, které využívá Bootstrap.

V konferenčním systému je pro definici designu a uživatelského rozhraní částečně využívána sada nástrojů Bootstrap doinstalovaná do projektu jako NuGet balíček *bootstrap* ve verzi 3.3.7. Bootstrap definuje sadu CSS tříd, které lze přiřazovat prvkům v pohledech

a tím rychle definovat jejich vzhled a případně i chování. [55] S využitím Bootstrap bylo v informačním systému vytvořeno například responzivní horní menu aplikace.

Aby se zdrojový kód prvků, které jsou na všech stránkách (například horní menu) nemusel opakovat v každém zdrojovém souboru pohledu, je v konferenčním systému vytvořena jedna hlavní či vzorová stránka (anglicky „Master Page“), kterou lze identifikovat pomocí její koncovky *.dotmaster*. Všechny pohledy se poté vkládají do této hlavní stránky na konkrétní místo, které je určeno DotVVM prvkem *<dot:ContentPlaceHolder ... />*. Stejně jako pohledy mají svůj „ViewModel“, i k hlavní stránce je přiřazen její „ViewModel“. Od tohoto hlavního „ViewModel“ všechny ostatní „ViewModels“ dědí. Každá stránka tak může přistoupit k vlastnosti, která je definována v hlavním „ViewModel“.

Zatímco v předchozích vrstvách byl jediným programovacím jazykem C#, v prezentační vrstvě již je kombinace jazyků bohatší. Pro vytvoření pohledů je využíván značkovací jazyk HTML v kombinaci s CSS a také speciální kód, který definuje DotVVM pro vytváření webových prvků, a případně JavaScript, i když v DotVVM využívání klientských skriptů v programovacím jazyku JavaScript, které by napsal sám programátor, není typické.

V projektu je využit rámec DotVVM, nainstalovaný jako NuGet balíček *DotVVM* ve verzi 1.1.7. DotVVM rozšiřuje množinu klíčových slov zdrojového kódu pohledu o své popisy webových prvků. Webové prvky DotVVM jsou uvozeny klíčovým slovem *dot:*. Mimo tento identifikátor se zapisují jako klasická HTML značka s atributy. Programátor má na výběr celou řadu předdefinovaných webových prvků, které stačí použít. Níže jsou uvedeny 3 ukázky. K názvu kontrolky je doplněn také zdrojový kód DotVVM, který ji definuje a následně je uvedeno, na jaký HTML kód se daná webová kontrolka renderuje: [64]

Button

– DotVVM:

```
<dot:Button Click="{command: Calculate()}" Text="Text" />
```

– Vyrenderované HTML na základě DotVVM webové kontrolky:

```
<input type="submit" onclick="dotvvm.postBack(...);" value="Text" />
```

nebo (na základě atributu *ButtonTagName*)

```
<button type="submit" onclick="dotvvm.postBack(...);">
       Text
</button>
```

CheckBox

– DotVVM:

```
<dot:CheckBox Text="CheckBox Label" Checked="{value: Checked}" />
```

– Vyrenderované HTML na základě DotVVM webové kontrolky:

```
<label>
  <input type="checkbox" data-bind="..." />
  CheckBox Label
</label>
```

Repeater

– DotVVM:

```
<dot:Repeater DataSource="{value: Items}">
  <ItemTemplate>
```

```
    {{value: _this}}<br />
  </ItemTemplate>
</dot:Repeater>
```

– Vyrenderované HTML na základě DotVVM webové kontrolky:

```
<!-- mod renderovani na klinetovi -->
<div data-bind="foreach: ...">
    <p><span data-bind="..."></span></p>
</div>

<!-- mod renderovani na serveru -->
<div>
    <p>Jim Hacker</p>
    <p>Humphrey Appleby</p>
    <p>Bernard Woolley</p>
</div>
```

DotVVM webové kontrolky poskytují navázání dat z „ViewModel“ do pohledu. V předchozích ukázkách je vidět, že toho lze ve zdrojovém kódu docílit pomocí složených závorek. Toto dynamické navázání hodnot je výhodné proto, že rámec DotVVM zajišťuje automatickou aktualizaci pohledu při změně hodnot ve „ViewModel“. Stránka se tak chová dynamicky a není vždy nutné při změně dat aktualizovat stránku.

Po vytvoření pohledů a „ViewModels“ zbývá poslední důležitý krok pro dokončení vývoje prezentační vrstvy, bez něhož by webová aplikace nefungovala. V třídě *DotvvmStartup*, která slouží pro spuštění DotVVM funkcionalit v projektu, je třeba v metodě *ConfigureRoutes* nastavit cesty k jednotlivým pohledům. Tím vznikne provázání mezi URL v prohlížeči a konkrétním pohledem, který je načten při přejití na danou URL.

5.4 Testy řízený vývoj

Testy řízený vývoj (anglicky „Test-Driven Development“, zkráceně TDD) je jedna z agilních metodik pro vývoj softwaru. Její základní princip spočívá v programování aplikace na základě neúspěšných automatických jednotkových testů.

V tradičním přístupu k vývoji softwaru jsou automatické testy mnohdy také využívány. Slouží ke zpětnému ověření, zda funkcionalita softwaru odpovídá specifikaci. V TDD se však posloupnost vytvoření testů a funkcionality aplikace obrací. Nejdříve jsou vytvořeny testy, které definují funkcionalitu aplikace. Teprve poté je implementována funkcionalita, u níž došlo k selhání odpovídajícího testu. Jednotkové testy tak nejsou pouze nástrojem kontroly, ale stávají se nástrojem pro návrh softwaru.

Vývojový cyklus založený na TDD lze rozdělit do následujících kroků: [8]

1. **Napsání testu**: Implementace každé funkcionality začíná napsáním testu, který výslednou funkcionalitu testuje, ověřuje správnost implementace a zároveň slouží jako definice požadavků.

2. **Spuštění testů a ujištění se, že žádný z nich neprojde**: Po napsání testů následuje jejich spuštění, které by mělo skončit neúspěchem. Žádný test by neměl projít, jelikož ještě není požadovaná funkcionalita implementována. Pokud by některý test prošel, je třeba se ujistit, zda test není chybný nebo zda již požadovaná funkcionalita není implementována v některém z přechozích cyklů vývoje.

3. **Napsání kódu (funkcionality aplikace)**: Teprve ve třetím kroku je implementována požadovaná funkcionalita, a to zejména s cílem, aby testy prošly. Na efektivitu a eleganci kódu není v tomto kroku kladen velký důraz.

4. **Spuštění testů a zjištění, zda všechny projdou**:Teprve ve třetím kroku je implementována požadovaná funkcionalita, a to zejména s cílem, aby prošly testy. Na efektivitu a eleganci kódu není v tomto kroku kladen velký důraz.

5. **Refaktorizace**: Předmětem posledního kroku – refaktorizace – je vytvoření efektivního a elegantního kódu. Po provedení refaktorizace je třeba opět spustit testy a ujistit se, že nedošlo k porušení výsledné funkčnosti.

Tento celý postup je opakován pro každou funkcionalitu aplikace.

Výhodou TDD je možnost opakovaně po každé změně v programu testovat všechny jeho funkcionality a požadavky. To vývojář může ocenit v případě, kdy dojde ke změně specifikace. Je třeba vytvořit nové testy nebo upravit několik stávajících a novou funkcionalitu implementovat. Poté je možné ověřit, že zásahem do kódu nedošlo ke zničení funkčnosti jiné části aplikace. Další výhodou je přesné definování požadavků pomocí testů. Tím nemůže při implementaci dojít k odchýlení se od specifikace, protože by poté neprošel test.

Tento agilní přístup má i své omezení a nevýhody. Využívání TDD vede k napsání většího množství kódu. Také je třeba myslet na to, že i v samotných testech se programátor může dopustit chyby. A ne každou část aplikace lze automaticky testovat. Například plnohodnotné automatické testování uživatelského rozhraní je velmi problematické, někdy až nemožné.

V konferenčním systému byl aplikován TDD při implementaci aplikační logiky (konkrétně pro implementaci služeb). Tato vrstva byla vybrána jako nejvhodnější pro vyzkoušení TDD, jelikož je možné ji nejvíce automaticky testovat, je možné definovat jaké návratové hodnoty aplikace očekává a je možné případné změny vracet do původního stavu. Kdyby s využitím TDD byla implementována například prezentační vrstva, vedlo by to k nutnosti složitého testování uživatelského rozhraní.

Služby představují logiku aplikace, kterou využívá prezentační vrstva pro práci s daty. Metody ze služeb lze rozdělit na metody, které čtou data z databáze a zpracovávají je před poskytnutím prezentační vrstvě (například je mapující mezi objekty datové vrstvy a DTOs) a metody, které data zpracovávají (například mapují v obráceném směru) a ukládají. Metody, které získávají data z databáze jsou většinou testovány, zda nevrací hodnotu „null“ a zda vrací očekávané hodnoty (minimálně výchozí údaje, které jsou do databáze vloženy při nasazení aplikace). Metody ukládající nové záznamy do databáze jsou testovány tak, že je v testu vytvořen zkušební objekt, který je následně uložen do databáze prostřednictvím aktuálně testované metody. Následně se zjišťuje, zda tento nový záznam v databázi opravdu vznikl a zda jeho vlastnosti odpovídají vlastnostem zkušebního objektu. Po testu je ještě třeba vrátit databázi do původního stavu například odstraněním zkušebního objektu. Obdobně jako metody pro přidávání záznamů do databáze fungují i metody upravující nebo odstraňující záznamy.

Kapitola 6

Nasazení a provoz aplikace

6.1 Provoz aplikace v testovacím prostřední

Kromě nasazení aplikace u zákazníka, je aplikace také dostupná ve své ukázkové verzi na adrese **https://konferencnisystem.azurewebsites.net**.

Tabulka 6.1 obsahuje přihlašovací údaje 3 různých uživatelů ve 3 různých rolích pro možnost vyzkoušení konferenčního systému:

Přihlašovací jméno (Email)	Heslo	Role
admin	KonfSysAdmin123	Administrátor
poradatel	KonfSysPoradatel123	Pořadatel konference
uzivatel	KonfSysUzivatel123	Účastník konference

Tabulka 6.1: Přihlašovací údaje ukázkových účtů a jejich role

Každý uživatel má jistá oprávnění dle své role. Níže jsou sepsána oprávnění pro konkrétní role.

Účastník konference:

- Objednat stravování na konferenci pro svoji osobu.
- Registrovat se na workshopy a přednášky.

Pořadatel konference:

- Zobrazit zaregistrované účastníky konference a jejich údaje.
- Upravit údaje účastníka zadané při registraci. Doplnit údaje, které není možné zadat při registraci (např. poznámka pořadatele).
- Zobrazit počty porcí a seznam strávníků k jednotlivým položkám ve stravování.
- Zobrazit a upravit, jaké jídlo má kterýkoliv účastník konference objednané.

- Zobrazit počet zaregistrovaných lidí a počet volných míst u každého workshopu nebo přednášky.
- Zobrazit a upravit, jaké workshopy nebo přednášky má kterýkoliv účastník konference zaregistrované.
- Exportovat jakékoliv výše zmíněné údaje do souboru ve formátu xslx.

Administrátor:

- Veškeré oprávněné, které má pořadatel konference.
- Právo smazat účastníka konference.

Veškeré nastavení oprávnění vychází ze specifikace konferenčního systémy a přesně odráží požadavky Zámeckého návrší, p.o.

6.2 Návod na nasazení a první spuštění aplikace

Konferenční systém je připraven na nasazení, spuštění a provozování na libovolné aplikační službě (anglicky „App Service“) v Microsoft Azure nebo lokálním serveru IIS Express, případně IIS. V aplikace je připraveno automatické vytvoření databáze a také její naplnění ukázkovými údaji včetně přidání tři základních účtů představených v tabulce 6.1.

Pro spuštění aplikace lokálně na serveru IIS Express je třeba postupovat dle návodu. Obdobný postup lze použít i pro nasazení na hosting Microsoft Azure:

1. Otevřít projekt ve vývojovém prostředí Visual Studio.
2. Nastavit připojovací řetězec k databázi do souborů *web.config* u projektů *ConferencySystem.DAL* a *ConferencySystemApp*. Připojovací řetězec může směřovat na lokální databázi MSSQL i na SQL databázi v Azure. Není třeba, aby databáze byla předem jakkoliv inicializována. Vytvoření tabulek a vložení výchozích dat zařídí aplikace při svém startu.
3. Jestliže již tabulky v databázi jsou vytvořeny a aplikace se k nim má pouze připojit, je třeba ve zdrojovém kódu zakomentovat inicializaci databáze. Tento scénář bude pravděpodobně potřeba jenom v případě aktualizace aplikace na novější verzi s přáním zachovat stejnou databázi. Inicializace databáze se vypne zakomentování řádku *Database.SetInitializer(new DbInitializer());* ve zdrojovém souboru *ConferencySystem.DAL/Data/DbContext.cs*.
4. Přeložit a spustit aplikaci (například pomocí klávesové zkratky *Ctrl + F5*).
5. Aplikace je připravena k použití. Stačí se přihlásit jedním ze tří účtů vytvořených při inicializaci databáze, které jsou uvedeny v tabulce 6.1.

V předcházejícím návodu bylo zmíněno, že vytvoření tabulek a vložení prvotních dat zajistí aplikace při svém prvním startu. Z toho důvodu může první spuštění aplikace po jejím úspěšném přeložení trvat až desítky sekund! Nejdříve je volán inicializer, který vytvoří prázdné tabulky v databázi. Následně je automaticky proveden první dotaz, který zapříčiní volání metody *Seed()* pro naplnění tabulek prvotními daty. Poté je aplikace již připravena k použití.

6.3 Nasazení a provoz aplikace u zákazníka

Způsob nasazení a spuštění aplikace v produkční verzi odpovídá postupu z kapitoly 6.2 Návod na nasazení a první spuštění aplikace. Konferenční systém je hostován v Microsoft Azure, stejně tak jako SQL databáze, která je k aplikaci připojena.

V produkční verzi jsou vzhledem ke zpracovávání osobních údajů zvýšené bezpečnostní požadavky na informační systém.

Dnes se již stává standardem komunikace přes protokol HTTPS (anglicky „Hypertext Transfer Protocol Secure“) namísto HTTP (anglicky „Hypertext Transfer Protocol“). HTTPS využívá pro komunikaci mezi klientem a serverem protokol HTTP spolu s protokolem SSL nebo TLS. Pomocí asymetrické kryptografie je ověřena identita webového serveru, poté následuje dohoda na klíči pro symetrické šifrování samotné komunikace. V případě, že by se útočníkovi podařilo zachytit komunikaci mezi klientem a serverem, získá pouze nečitelná zašifrovaná data, která bez šifrovacího klíče budou bezcenná. [87]

V konferenčním systému je využíván protokol HTTPS namísto HTTP. I při pokusu o přístup k webové aplikaci přes protokol HTTP dojde k přesměrování a další komunikace probíhá přes protokol HTTPS. Díky tomu je veškerá komunikace zabezpečená a důvěryhodná. Například rizikové odeslání osobních údajů poskytnutých při registraci z klienta na server je šifrováno. A protokol HTTPS také zajišťuje důvěryhodnost přenášených dat například při zaslání informací o účastnících konference ze serveru do klientského prohlížeče.

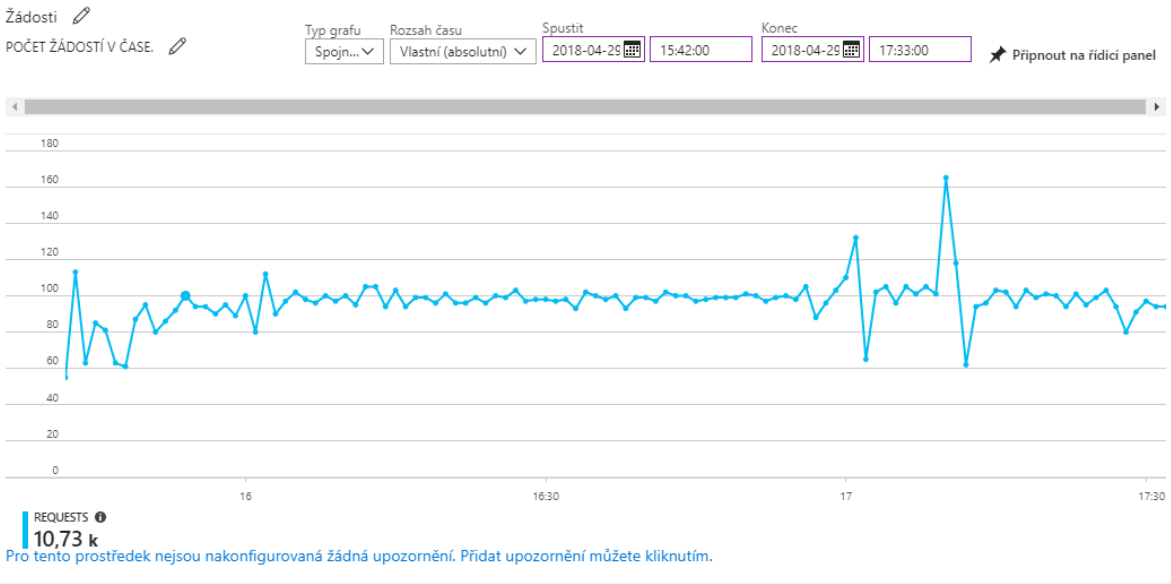
Pro protokoly SSL a TLS je zásadní infrastruktura veřejného klíče a X.509 certifikáty, díky nimž probíhá autentizace. V konferenčním systému je certifikát zasílaný klientovi zprostředkován certifikační autoritou „Microsoft IT TLS CA 4“. [87]

Kromě zabezpečení dat při síťové komunikaci, jsou veškeré údaje v konferenčním systému chráněny i na úrovni databáze. U SQL databáze v Microsoft Azure je aktivováno zabezpečení technologií „Transparent data encryption“, která poskytuje šifrování a dešifrování databáze v reálném čase. V konferenčním systému je pro šifrování databáze využívána symetrická kryptografie. Při požadavku na načtení dat z databáze dojde k jejich rozšifrování, načtení a zaslání aplikaci. Při požadavku na uložení dat je proces obrácený. [37] [93] Díky šifrování databáze je možné data chránit proti útoku cílenému přímo proti databázi s úmyslem získání důvěrných údajů. I v případě úspěšného útoku by hacker získal pouze šifrovaná data, která jsou bez šifrovacího klíče opět bezcenná.

Dalším důležitým aspektem produkční verze aplikace je její rychlost. Rychlost aplikace se dá přesně měřit zjištěním průměrné doby obsluhy jednoho požadavku. Tento údaj říká, jak dlouho musí klient průměrně čekat na odpověď serveru při zaslání jednoho požadavku.

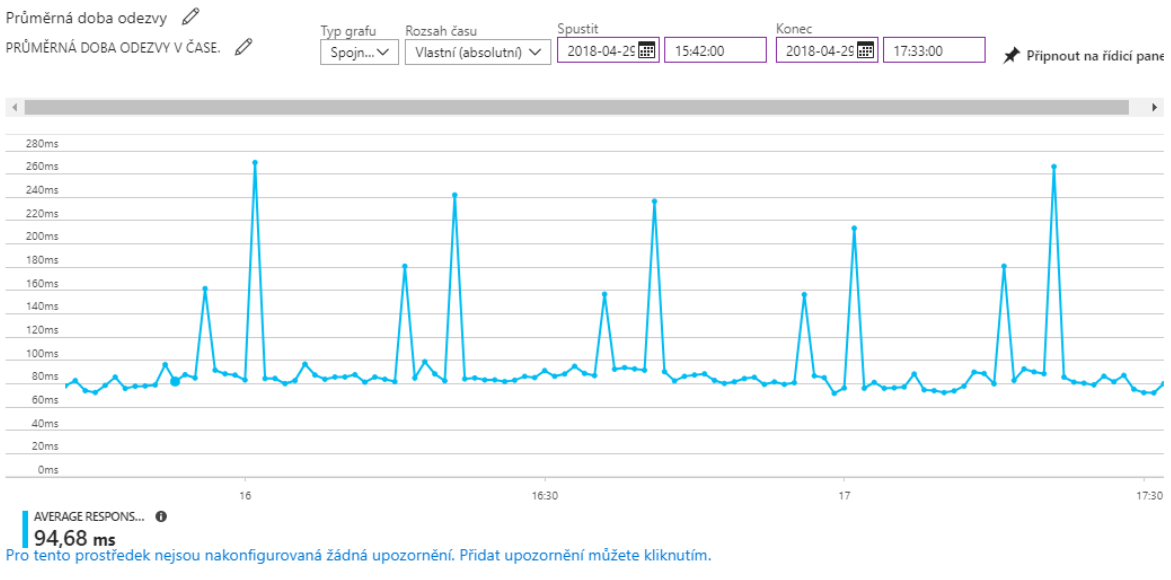
Při intenzivním testovacím provozu byly po dobu 1 hodiny a 47 minut zaznamenávány požadavky na straně serveru. Z těchto dat jsou vytvořeny grafy níže. Měření probíhalo pomocí metrik implementovaných přímo v hostovacím prostředí Microsoft Azure.

První graf 6.1 znázorňuje počet požadavků na serverovou část aplikace za čas. Za zkoumanou dobu (1 hodina a 47 minut) server obsloužil celkem přibližně 10 730 požadavků. Nejedná se o maximální počet požadavků, které je server schopný obsloužit, ale o počet požadavků, které vznikly při testovacím provozu aplikace. Maximální počet obsloužených požadavků je dle dokumentu „Azure subscription and service limits, quotas, and constraints“ 20 000 požadavků za sekundu. [70] K tomuto číslu se běžný provoz aplikace ani při velké zátěži neblíží, tudíž lze hosting Microsoft Azure považovat za dostatečně výkonný z hlediska počtu obsloužených požadavků za sekundu.



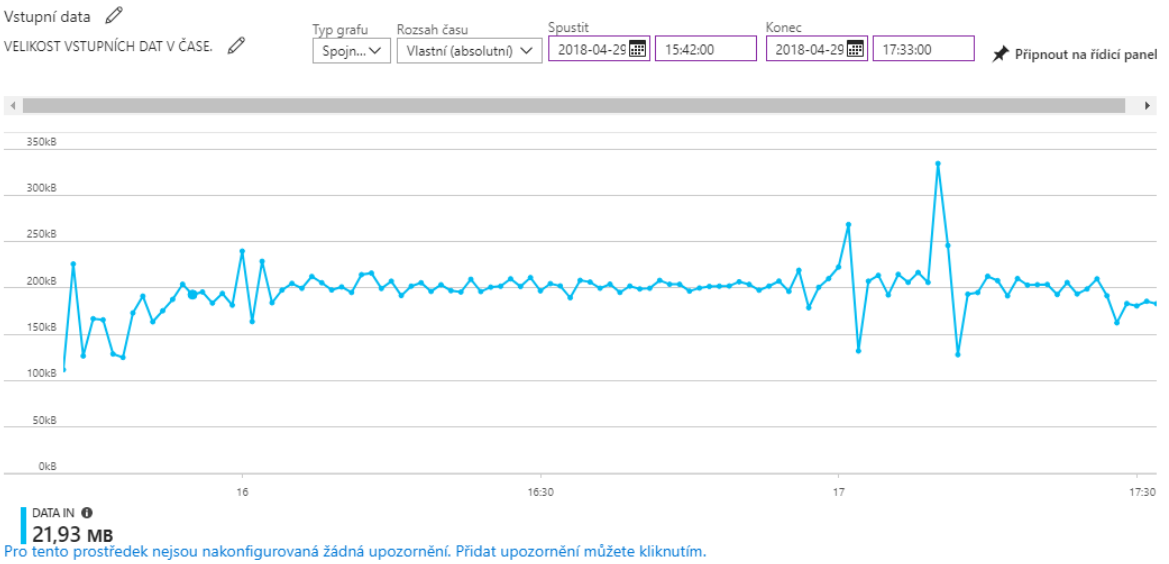
Obrázek 6.1: Počet požadavků na serverovou část webové aplikace v čase

Druhý graf 6.2 je nejdůležitější pro odpověď na otázku, zda je serverová část webové aplikace dostatečně výkonná. Zobrazuje průměrnou dobu odezvy – tedy dobu potřebnou na vyřízení jednoho požadavku. Za sledovaný čas trvalo obsloužení jednoho požadavku v průměru 94,68 ms. Jaká je maximální možná průměrná doba odezvy, aby aplikace z pohledu klienta byla plynulá, nelze jednoznačně říci. Některé článku uvádějí, že zpracování jednoho požadavku na straně serveru by mělo být průměrně nižší než 500 ms, některé uvádějí až 200 ms. [11] [53] [52] [13] Vzhledem k získaným datům, kde průměrná doba potřebná pro obsloužení jednoho požadavku na straně serveru nedosáhla ani 100 ms, lze prohlásit serverovou část webové aplikace za dostatečně výkonnou.

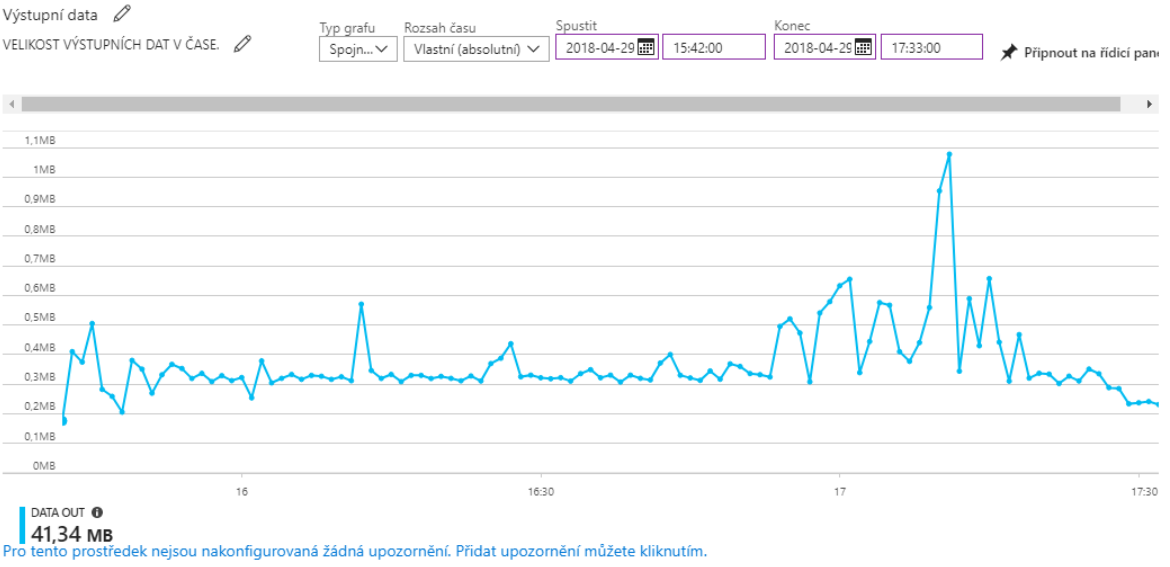


Obrázek 6.2: Průměrná doba odezvy v čase

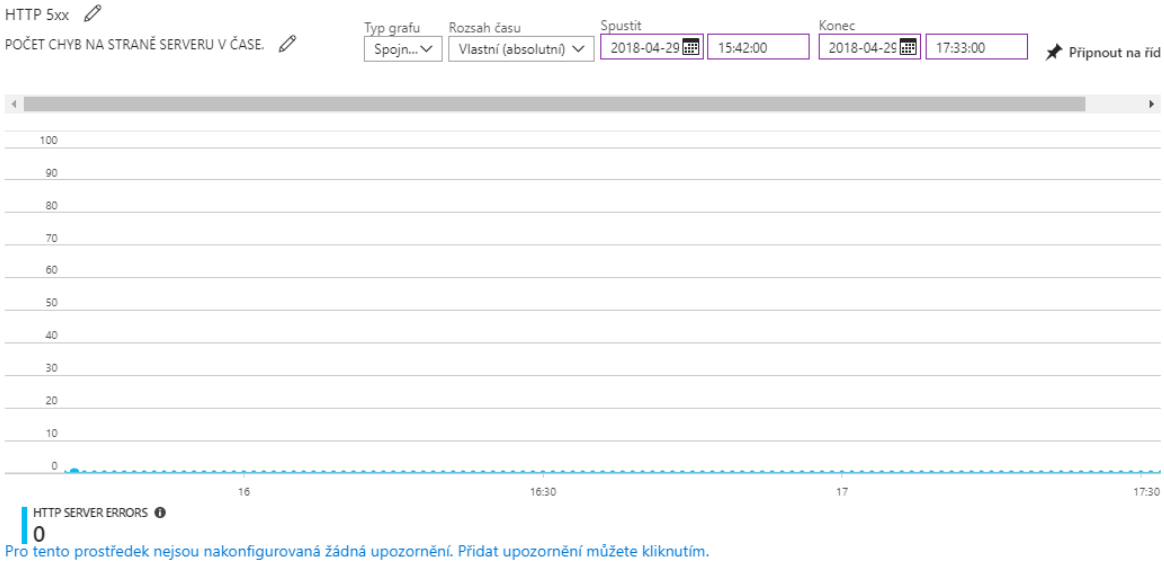
Následující tři grafy 6.3, 6.4 a 6.5 rozšiřují informační hodnotu měření o další hodnoty, které již sice nejsou klíčové z hlediska výkonu a rychlosti aplikace, ale slouží k ucelenému pohledu na aktivitu konferenčního systému. Graf 6.3 zobrazuje přenesené množství vstupních dat a graf 6.4 přenesené množství dat výstupních. Graf 6.5 pouze doplňuje informaci, kolikrát za zkušebního měřeného provozu došlo k chybě na straně serveru. Pozitivním výsledkem je, že nebyla zaznamenána jediná chyba.



Obrázek 6.3: Velikost vstupních dat v čase



Obrázek 6.4: Velikost výstupních dat v čase



Obrázek 6.5: Počet chyb na straně serveru v čase

Kapitola 7

Návrhy na další vývoj

Po nasazení konferenčního systému v produkční verzi došlo k fázi zhodnocení projektu se zadavatelem, jehož výsledem bylo, že webová aplikace splnila specifikaci v plném rozsahu a vyhovuje potřebám Zámeckého návrší, p.o. Ač požadavky byly splněny, stále existují zlepšení, které je možné v budoucím vývoji uskutečnit. Níže jsou sepsány ty nejpodstatnější.

V budoucnu očekávám pokračování spolupráce s neziskovou organizací Zámecké návrší, p.o. a tím i uskutečnění níže představených návrhů, které by mohly vést k vylepšení či rozšíření webové aplikace.

7.1 Získávání údajů o organizaci z ARES

Prvním návrh se týká registrace uživatelů. Při registraci je možné zadat také informace o organizaci, ke které účastník konference patří. Aktuálně uživatel vyplňuje manuálně veškeré údaje (IČ, DIČ, Název organizace, Ulice a číslo popisné, Město a PSČ). Pro vyplňování formuláře by bylo možné využít získávání dat z Administrativního registru ekonomických subjektů (zkráceně ARES) přes jeho API. ARES je webová aplikace Ministerstva financí, která souhrnně zpřístupňuje údaje z informačních systémů pro vedení registrů a evidencí veřejné správy o ekonomických subjektech. [94] S využitím získávání dat z ARES by uživateli stačilo zadat IČ, na jehož základě by byly doplněny zbývající údaje. Vedlo by to k eliminaci možných chyb při vyplňování formuláře ze strany uživatele.

7.2 Automatické testování grafického rozhraní

V konferenčním systémy aktuálně existují jednotkové testy ověřující funkčnost všech služeb, které tvoří aplikační logiku. V budoucnu by bylo možné rozšířit testy i na grafické uživatelské rozhraní (anglicky „Graphical Users Interface“, zkráceně GUI), čímž by se dosáhlo kontroly, zda v uživatelském rozhraní neexistují nedostatky.

Testování uživatelského rozhraní slouží k identifikování chyb vzniklých v průběhu fáze tvorby vzhledu aplikace, k ověření funkčnosti grafického rozhraní (například testování vzniku události nebo odeslání formuláře po kliknutí na tlačítko). Dále GUI testování vyhodnocuje hodnoty na webových kontrolkách a dalších prvcích. Pokud jsou například do rozbalovacího menu doplňovány prvky dynamicky na základě načtených dat, testy GUI jsou schopné vyhodnotit jaké prvky bude ve výsledku obsahovat dané rozbalovací menu a po vyhodnocení může kontrolovat jejich validitu. Kromě vyhodnocení dynamicky doplňovaných hodnot

v grafickém rozhraní lze provádět v testu akce nad objekty jako je kliknutí, najetí kurzorem na objekt apod. [27]

Níže jsou příklady, co může být ověřováno při testech grafického uživatelského rozhraní:

- Ověření dosažitelnosti pohledů
- Velikost a pozice webových kontrolek a dalších prvků
- Dobré zarovnání prvků (například obrázků)
- Funkčnost navigace a odkazů
- Font, formátování a zarovnání textů
- Výsledné hodnoty textových polí
- Zobrazení chybových zpráv
- Webové prvky pro zobrazování průběhu (anglicky „Progress Bars“)

Automatické testování uživatelského rozhraní by přineslo rychlou možnost ověření správné funkčnosti této nejvrchnější vrstvy aplikace, která slouží pro interakci s uživatelem. Zejména po změnách nejen v prezentační vrstvě by bylo možné rychle ověřit, zda nedošlo při úpravách k chybě a narušení funkčnosti některého prvku.

7.3 Nový design

Aktuální design aplikace odpovídá požadavkům na srozumitelné uživatelské rozhraní poskytující intuitivní práci s webovou aplikací. Například pravidla pro barvy, kde červená znamená chybu, zelená naopak úspěšné dokončení operace, jsou v konferenčním systému striktně dodržována. Také rozvržení prvků webu odpovídá zvyklostem. Hlavní menu je v horní části webové stránky a je stejné na všech pohledech, Tlačítko pro uložení se nachází pod formulářem, chybové hlášky se zobrazují nad ostatním obsahem webu apod.

Přesto ve vzhledu aplikace je prostor pro modernizaci. Místo stylů z Bootstrap by bylo možné v budoucnu napsat vlastní styly charakteristické pouze pro konferenční systém Zámeckého návrší, p.o. Velkým přínosem z pohledu propagace značky Zámecké návrší by byla aplikace grafických pravidel organizace na konferenční systém. Ve specifikaci tento požadavek nebyl, přesto z hlediska marketingu by pro organizaci bylo jistě přínosné vytvoření vzhledu s využitím firemních barev, fontů, formátování a obrázků.

Při tvorbě nového designu by byla potřebná spolupráce s designerem a grafikem této organizace.

7.4 Refaktorizace kódu

Konferenční systém zajišťuje aktuálně veškerou potřebnou funkcionalitu, což potvrzují i úspěšně proběhlé všechny testy. Přesto v budoucnu se nabízí provedení refaktorizace a úpravy kódu. Při postupném vývoji vznikly některé duplicitní metody či metody s podobnou funkcionalitou. Částečná refaktorizace již byla provedena. Většinou však pouze v rámci jedné třídy. Při komplexnějším pohledu na zdrojové kódy konferenčního systému je tak možné odhalit další možnosti refaktorizace.

Díky aplikaci TDD a napsání testů, které kompletně pokrývají logiku aplikace, by refaktorizace byla snazší na provedení oproti případu, kdy by testy neexistovaly. Nyní je možné po každé změně spustit testy a ujistit se, že funkčnost definovaná ve specifikaci je plně zachována i po změnách.

Kapitola 8

Závěr

Cílem bakalářské práce bylo vytvoření konferenčního informačního systému pro neziskovou organizaci Zámeckého návrší, p.o. Dílčí cíle vedoucí ke vzniku této webové aplikace zahrnovaly seznámení se s platformou .NET, nastudování a vyzkoušení webových rámců a technologií nad platformou .NET se zaměřením se na nový rámec DotVVM; dále analýzu požadavků na konferenční systém a návrh aplikace na základě specifikace. Závěrečnými cíli bylo na základě specifikace a návrhu implementovat výslednou webovou aplikaci s využitím testy řízeného vývoje i pro grafické uživatelské rozhraní a nasadit ji do produkčního prostředí, otestovat v reálném provozu a učinit návrhy na další možný vývoj.

Cíl práce se podařilo splnit a jejím výsledkem je plně funkční konferenční informační systém nasazený jak v produkčním prostředí pro Zámecké návrší, p.o., tak v ukázkovém prostředí (https://konferencnisystem.azurewebsites.net), kde je možné webovou aplikaci vyzkoušet. Dílčích cílů bylo také úspěšně dosaženo s částečnou výjimkou testy řízeného vývoje, který byl limitován pouze na logiku aplikace a nebyl využit při tvorbě grafické uživatelské rozhraní. Toto omezení cíle proběhlo vzhledem k časové náročnosti problematiky a s ohledem na relativně nekomplikované uživatelské rozhraní, kde je efektivnější provést vyzkoušení funkčnosti webových prvků manuálně nežli vytvářet testy. Automatické testování grafického uživatelského rozhraní však je zahrnuto do návrhů na další vývoj aplikace společně s vytvořením nového vzhledu aplikace.

Přínosem pro odbornou veřejnost je analýza nového webového rámce DotVVM v porovnání s dalšími webovými rámci nad platformou .NET. K této problematice dosud nebyla vydána komplexnější publikace, informace bylo možné získávat pouze z vlastní zkušenosti či z článků, které se vždy zaměřovaly pouze na konkrétní části problematiky. První část práce zabývající se popsáním různých přístupů při webovém vývoji nad platformou .NET může sloužit vývojářům jako důvěryhodné vodítko podložené mnoha prameny pro výběr implementačních technologií.

Přínos práce je nejvýraznější pro neziskovou organizaci Zámecké návrší, p.o., která předtím nevyužívala žádný podobný systém a veškeré procesy byly řešeny pomocí Formulářů Google s velmi omezenou funkčností oproti konferenčnímu systému.

Při vývoji konferenčního systému bylo plně dosaženo požadavků ve specifikaci a aktuálně je webová aplikace již nasazena v produkčním prostředí. V budoucnu je očekávána další spolupráce s organizací a pokračování ve vývoji aplikace. K urychlení vyplňování registračního formuláře a k eliminaci chyb ze strany uživatele by vedlo získávání údajů o organizaci přes API webové aplikace ARES na základě IČ organizace. Dalším pokračováním ve vývoji by bylo rozšíření automatických testů z logiky aplikace i na její grafické uživatelské rozhraní.

Tento krok by bylo vhodné spojit s vytvořením nového designu a částečnou refaktorizací zdrojového kódu aplikace.

Námětem vycházející ze zkušeností s tvorbou konferečního systému je důležitost výběru vhodné implementační technologie. Ač univerzálním řešením pro webový vývoj může být nad platformou .NET rámec ASP.NET MVC, je často velmi vhodné věnovat čas pro zvážení jiného rámce i za cenu nutnosti se na začátku naučit novou technologii. Pomocí rámce, vytvořeného pro specifická řešení, lze dosáhnout nejen rychlejšího a efektivnějšího vývoje, ale také optimalizace výkonu aplikace či vyšší bezpečnosti. Při výběru vhodné implementační technologie může pomoci i tato práce, která se mimo jiné zabývá analýzou rámců včetně vytvoření ukázkových aplikací a zhodnocení vhodnosti daného rámce pro různé druhy webových aplikací. Výsledný konferenční informační systém je ukázkou vývoje webové aplikace, ve které je kladen velký důraz na bezpečnost a vhodnou architekturu, při zvolení rámce DotVVM.

Literatura

[1] Anderson, R.; Addie, S.; Latham, L.; aj.: *Microsoft Ajax Content Delivery Network*. Microsoft, Říjen 2017, [Online; navštíveno 19.01.2018].
URL https://docs.microsoft.com/en-us/aspnet/ajax/cdn/overview

[2] Anderson, R.; Latham, L.; Mullen, T.; aj.: *Razor syntax reference for ASP.NET Core*. Microsoft, Říjen 2017, [Online; navštíveno 10.01.2018].
URL https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/razor

[3] Anderson, R.; Pasic, A.; Dykstra, T.: *What is Web Forms*. Microsoft, Únor 2014, [Online; navštíveno 10.12.2017].
URL https://docs.microsoft.com/en-us/aspnet/web-forms/what-is-web-forms

[4] archil: *Will ASP.NET MVC replace ASP.NET Web Forms in future?* StackOverflow, Únor 2013, [Online; navštíveno 15.12.2017].
URL https://stackoverflow.com/questions/6530349/will-asp-net-mvc-replace-asp-net-web-forms-in-future

[5] Barth, A.: *HTTP State Management Mechanism*. RFC 2965, RFC Editor, April 2011, [Online; navštíveno 20.01.2018].
URL https://tools.ietf.org/rfc/rfc6265.txt

[6] Basques, K.: *Get Started with Analyzing Network Performance in Chrome DevTools*. Tools for Web Developers, Leden 2018, [Online; navštíveno 19.01.2018].
URL https://developers.google.com/web/tools/chrome-devtools/network-performance/

[7] Bass, L.; Clements, P.; Kazman, R.: *Software architecture in practice*. Boston: Addison-Wesley, první vydání, 2003, ISBN 0-321-15495-9.

[8] Beck, K.: *Programování řízené testy*. U Průhonu 22, Praha 7: Grada Publishing, první vydání, 2004, ISBN 80-247-0901-5.

[9] Bruckner, T.; Voříšek, J.; Buchalcevová, A.; aj.: *Tvorba informačních systémů: Principy, metodiky, architektury*. U Průhonu 22, Praha 7: GRADA Publishing, první vydání, 2012, ISBN 978-80-247-4153-6.

[10] Burns, A.; Dunn, C.; aj.: *Introduction to Mobile Development*. Microsoft, Březen 2017, [Online; navštíveno 2.12.2017].
URL https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/introduction-to-mobile-development

[11] Carlson, J.: *How to Reduce Server Response Times*. Rigor, Září 2016, [Online; navštíveno 29.04.2018].
URL https://rigor.com/blog/2016/09/how-to-reduce-server-response-times

[12] Chauhan, S.: *What is Web API and why to use it ?* DotNetTricks, Září 2016, [Online; navštíveno 18.01.2018].
URL http://www.dotnettricks.com/learn/webapi/what-is-web-api-and-why-to-use-it-

[13] Cherouvim, I.: *What is considered average request processing time?* StackOverflow, Březen 2009, [Online; navštíveno 29.04.2018].
URL https://stackoverflow.com/questions/628229/what-is-considered-average-request-processing-time

[14] Drayton, P.; Albahari, B.; Neward, T.: *C# v kostce*. U Průhonu 22/466, Praha 7, Holešovice: Grada Publishing a.s., první vydání, 2003, ISBN 80-247-0443-9.

[15] Duthie, G. A.: *ASP.NET krok za krokem*. Jundrovská 33, 624 00 Brno: Mobil Media a.s., první vydání, 2003, ISBN 80-86593-33-9.

[16] ECMA International: *Standard ECMA-335 - Common Language Infrastructure (CLI)*. Geneva, Switzerland, 6 vydání, Červen 2012, [Online; navštíveno 2.12.2017].
URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf

[17] Esposito, D.: Which ASP.NET Is Better? *Information Week*, ročník 1307, č. 8, 2011: s. 50–52, ISSN 8750-6874, doi:https://search.proquest.com/docview/893819787/fulltextPDF/A353D295A57E4C84PQ/1?accountid=17115, [Online; navštíveno 15.12.2017].

[18] parlament a rada EU, E.: *NAŘÍZENÍ EVROPSKÉHO PARLAMENTU A RADY (EU) 2016/679 ze dne 27. dubna 2016 o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů a o zrušení směrnice 95/46/ES (obecné nařízení o ochraně osobních údajů)*. http://eur-lex.europa.eu, Duben 2016, [Online; navštíveno 20.02.2018].
URL http://eur-lex.europa.eu/legal-content/CS/TXT/HTML/?uri=CELEX:32016R0679&from=EN

[19] FitzMacken, T.; Addie, S.; Pasic, A.; aj.: *Programming ASP.NET Web Pages (Razor) Using Visual Studio*. Microsoft, Únor 2014, [Online; navštíveno 10.01.2018].
URL https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/program-asp-net-web-pages-in-visual-studio

[20] Freeman, A.; Sanderson, S.: *Pro ASP.NET MVC 3 Framework*. New York: Apress, třetí vydání, 2011, ISBN 978-1-4302-3404-3.

[21] Galloway, J.; Addie, S.; Latham, L.; aj.: *Introduction to ASP.NET Identity*. Microsoft, Říjen 2013, [Online; navštíveno 10.03.2018].
URL https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

[22] Herceg, T.: *Náš rok s dotvvm*. dotNETportal, Duben 2016, [Online; navštíveno 16.01.2018].
URL
https://www.dotnetportal.cz/blogy/3/Tomas-Herceg/8494/Nas-rok-s-DotVVM

[23] Herceg, T.: *Visual studio 2017, .net core a nový formát projektů*. dotNETportal.cz, Prosinec 2016, [Online; navštíveno 6.12.2017].
URL https://www.dotnetportal.cz/clanek/8506/Visual-Studio-2017-NET-Core-a-novy-format-projektu

[24] Herceg, T.: *Modernizing ASP.NET Web Forms Applications (Part 1)*. Tomáš Herceg, Listopad 2017, [Online; navštíveno 15.12.2018].
URL https://tomasherceg.com/blog/post/modernizing-asp-net-web-forms-applications-part-1

[25] Herceg, T.: *We are RIGANTI!* Riganti, Listopad 2017, [Online; navštíveno 15.01.2018].
URL https://www.riganti.cz/en

[26] Herceg, T.; Jež, A.; Jurásek, T.; aj.: *Open source MVVM framework for Web Apps*. GitHub, Březen 2018, [Online; navštíveno 16.01.2018].
URL https://github.com/riganti/dotvvm

[27] Holmes, J.: *Getting Started with UI Test Automation*. redgate Hub, Listopad 2014, [Online; navštíveno 05.05.2018].
URL https://www.red-gate.com/simple-talk/dotnet/asp-net/getting-started-with-ui-test-automation/

[28] Homer, A.: *Professional ASP.NET web forms techniques*. Birmingham: Wrox Press, první vydání, 2002, ISBN 1-86100-786-8.

[29] JetBrains: *ReSharper*. JetBrains, Říjen 2017, [Online; navštíveno 10.02.2018].
URL https://www.jetbrains.com/resharper/

[30] Jones, M.; Wenzel, M.; Latham, L.; aj.: *Language independence and language-independent components*. Microsoft, Červenec 2016, [Online; navštíveno 24.11.2017].
URL
https://docs.microsoft.com/en-us/dotnet/standard/language-independence

[31] Jurásek, T.: *Introduction To DotVVM*. C#Corner, Březen 2018, [Online; navštíveno 15.01.2018].
URL https://www.c-sharpcorner.com/article/introduction-to-dotvvm/

[32] Koirala, S.: *Webforms vs MVC and Why MVC is better ?* Code Project, Září 2014, [Online; navštíveno 10.12.2017].
URL https://www.codeproject.com/Articles/821275/Webforms-vs-MVC-and-Why-MVC-is-better

[33] Kubíček, I.: *Firmy stále nejsou na GDPR připraveny*. Hospodářské noviny, Únor 2018, [Online; navštíveno 20.02.2018].
URL https:
//archiv.ihned.cz/c1-66048650-firmy-stale-nejsou-na-gdpr-pripraveny

[34] Kumar, R.: *Advantages and Disadvantages of using ASP.NET MVC Framework*. C# Corner, Červenec 2010, [Online; navštíveno 8.01.2018].
URL https://www.c-sharpcorner.com/blogs/advantages-and-disadvantages-of-using-asp-net-mvc-framework1

[35] Kurzy.cz: *Každá druhá firma nebude včas připravena na GDPR*. Kurzy.cz, Prosinec 2017, [Online; navštíveno 20.02.2018].
URL https://www.ekonomickymagazin.cz/2018/04/na-gdpr-nejsou-pripraveny-ctyri-petiny-tuzemskych-eshopu/

[36] LLC, U.: *UltiDev Cassini Web Server for ASP.NET Applications*. UltiDev, [Online; navštíveno 15.12.2018].
URL http://ultidev.com/products/Cassini/

[37] Macauley, E.; Guyer, C.; Milener, G.; aj.: *Transparent Data Encryption (TDE)*. Microsoft, Září 2017, [Online; navštíveno 20.04.2018].
URL https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=azuresqldb-current

[38] Mani, V.: *Advantage and Disadvantage of ASP.NET MVC*. C# Corner, Červen 2015, [Online; navštíveno 8.01.2018].
URL https://www.c-sharpcorner.com/blogs/advantage-and-disadvantage-of-asp-net-mvc1

[39] Margai, L.: *Využití návrhového vzoru Model-View-Controller ve webových aplikacích*. Bakalářská práce, Vysoké Učení Technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Antonínská 548/1, 601 90 Brno, 2013.

[40] Marian, L.: *Architektura softwaru a návrhové vzory*. Diplomová práce, Univerzita Pardubice, Fakulta elektrotechniky a informatiky, Katedra softwarových technologií, Studentská 95, 532 10 Pardubice 2, 2014.

[41] Microsoft: *ASP.NET*. Microsoft, [Online; navštíveno 15.12.2018].
URL https://msdn.microsoft.com/cs-cz/library/dd566231(v=vs.110).aspx

[42] Microsoft: *Design Fundamentals*. Microsoft, Říjen 2009, [Online; navštíveno 26.02.2018].
URL https://msdn.microsoft.com/en-us/library/ee658116.aspx

[43] Microsoft: *Návod: vytvoření webu pomocí syntaxe Razor v aplikaci Visual Studio*. Microsoft, Březen 2010, [Online; navštíveno 10.01.2018].
URL https://msdn.microsoft.com/cs-cz/library/gg606533(v=vs.100).aspx

[44] Microsoft: *Chapter 6: Presentation Layer Guidelines*. Microsoft, Červenec 2015, [Online; navštíveno 05.04.2018].
URL https://msdn.microsoft.com/en-us/library/ee658081.aspx

[45] Microsoft: *System.Net.Mail Namespace*. Microsoft, Únor 2015, [Online; navštíveno 30.04.2018].
URL
https://msdn.microsoft.com/en-us/library/system.net.mail(v=vs.110).aspx

[46] Microsoft: *Dokumentace Entity Framework*. Microsoft, Leden 2017, [Online; navštíveno 10.03.2018].
URL https://docs.microsoft.com/cs-cz/ef/

[47] Microsoft: *Microsoft Azure*. Microsoft, Leden 2018, [Online; navštíveno 10.02.2018].
URL https://azure.microsoft.com/cs-cz/

[48] Microsoft: *Microsoft Docs*. Microsoft, Leden 2018, [Online; navštíveno 10.02.2018].
URL https://docs.microsoft.com/en-us/

[49] Microsoft: *NuGet Gallery | Packages*. Microsoft, Leden 2018, [Online; navštíveno 10.02.2018].
URL https://www.nuget.org/packages

[50] Microsoft: *Visual Studio*. Microsoft, Leden 2018, [Online; navštíveno 10.02.2018].
URL https://www.visualstudio.com/

[51] Müller, D.: *Síťová komunikace v .NET Framework*. Bakalářská práce, Vysoké Učení Technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Antonínská 548/1, 601 90 Brno, 2012.

[52] Network, Y. D.: *Best Practices for Speeding Up Your Web Site*. Yahoo! Developer Network, Duben 2010, [Online; navštíveno 29.04.2018].
URL https://developer.yahoo.com/performance/rules.html?guccounter=1

[53] Nielsen, J.: *Response Times: The 3 Important Limits*. Nielsen Norman Group, Únor 2014, [Online; navštíveno 29.04.2018].
URL https://www.nngroup.com/articles/response-times-3-important-limits/

[54] Noyes, B.: *Building ASP.NET Single Page Applications in HTML 5 with Upshot*. *Code Magazine*, , č. 5, 2012, ISSN 1547-5166, doi:http://www.codemag.com/article/1210061/Building-ASP.NET-Single-Page-Applications-in-HTML-5-with-Upshot, [Online; navštíveno 10.01.2018].

[55] Otto, M.; Rebert, C.; Artemchuk, V.; aj.: *Bootstrap*. GitHub, Duben 2018, [Online; navštíveno 10.04.2018].
URL https://github.com/twbs/bootstrap/blob/v4-dev/README.md

[56] Patil, B.: *Test Driven Development (TDD) Using MVC Web Application*. C# Corner, Březen 2016, [Online; navštíveno 10.12.2017].
URL https://www.c-sharpcorner.com/article/test-driven-development-tdd-using-mvc-web-application3/

[57] Petrusha, R.; Jones, M.; Hoffman, M.; aj.: *Building Console Applications in the .NET Framework*. Microsoft, Březen 2017, [Online; navštíveno 2.12.2017].
URL
https://docs.microsoft.com/en-us/dotnet/standard/building-console-apps

[58] Petrusha, R.; Jones, M.; Hoffmann, M.; aj.: *.NET Framework Class Library Overview*. Microsoft, Únor 2018, [Online; navštíveno 10.12.2017].
URL
https://docs.microsoft.com/en-us/dotnet/standard/class-library-overview

[59] Petrusha, R.; aj.: *Common Language Runtime (CLR)*. Microsoft, Říjen 2017, [Online; navštíveno 24.11.2017].
URL https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/EN-US.aspx

[60] Prabhu, N.: *Analyze POST and GET packets using WireShark*. nimishprabhu.com, Duben 2016, [Online; navštíveno 19.01.2018].
URL http://nimishprabhu.com/analyze-post-and-get-packets-using-wireshark.html

[61] Primakovski, B.: *17 Best Python Web Frameworks to Learn in 2017*. STEEL KIWI, Březen 2017, [Online; navštíveno 10.02.2018].
URL https://steelkiwi.com/blog/best-python-web-frameworks-to-learn/

[62] Prosise, J.: *Programování v Microsoft .NET*. Nám. 28. dubna 48, 635 00 Brno: Computer Press, první vydání, 2003, ISBN 80-7226-879-1.

[63] Protalinski, E.: *MIX09: Roundup of first keynote announcements*. ArsTechnica, Březen 2009, [Online; navštíveno 15.12.2018].
URL https://arstechnica.com/information-technology/2009/03/mix09-roundup-of-first-keynote-announcements/

[64] Riganti: *Controls: AuthenticatedView*. DotVVM, Leden 2017, [Online; navštíveno 09.04.2018].
URL https://msdn.microsoft.com/en-us/library/ee658081.aspx

[65] Riganti: *GridView*. Riganti, Březen 2017, [Online; navštíveno 16.01.2018].
URL https://www.dotvvm.com/docs/controls/builtin/GridView/latest

[66] Riganti: *InlineScript*. Riganti, Březen 2017, [Online; navštíveno 16.01.2018].
URL https://www.dotvvm.com/docs/controls/builtin/InlineScript/latest

[67] Riganti: *Introduction*. Riganti, Březen 2017, [Online; navštíveno 8.01.2018].
URL https://www.dotvvm.com/docs/tutorials/introduction/latest

[68] Robinson, S.; Allen, K.; Cornes, A.; aj.: *C# Programujeme profesionálně*. Nám. 28. dubna 48, 635 00 Brno: Computer Press, první vydání, 2003, ISBN 80-251-0085-5.

[69] Rogers, A.; Brewer, G.: *ASP.NET Usage Statistics - Websites using ASP.NET*. BuiltWith, Leden 2018, [Online; navštíveno 7.12.2017].
URL https://trends.builtwith.com/framework/ASP.NET

[70] Roth, J.; Deng, S.; aj.: *Azure subscription and service limits, quotas, and constraints*. Microsoft, Březen 2018, [Online; navštíveno 29.04.2018].
URL https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits

[71] Sahasrabuddhe, R.: *Developer Tools JavaScript Frameworks for ASP.NET MVC Developers*. DotNetCurry, Duben 2017, [Online; navštíveno 8.01.2018].
URL http://www.dotnetcurry.com/javascript/1359/javascript-frameworks-aspnet-mvc-developer

[72] Schneider, S.: *Single-Page vs. Multi-page UI Design: Pros & Cons*. Studio by UXPin, Říjen 2010, [Online; navštíveno 10.01.2018].
URL https://www.uxpin.com/studio/blog/single-page-vs-multi-page-ui-design-pros-cons/

[73] Skvorc, B.: *The Best PHP Framework for 2015: SitePoint Survey Results*. SitePoint, Březen 2015, [Online; navštíveno 10.02.2018].
URL https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[74] Sommerville, I.: *Softwarové inženýrství*. Brno: Computer Press, první vydání, 2013, ISBN 978-80-251-3826-7.

[75] Stehlik, C.: *Getting Started Guide*. GitHub, Říjen 2017, [Online; navštíveno 20.03.2018].
URL https://github.com/AutoMapper/AutoMapper/blob/877d917a2d2a20e8b6ab49d7436747a75a67e7b4/docs/Getting-started.md

[76] Sullivan, W.: *Benefits of MVVM over MVC*. StackOverflow, Říjen 2009, [Online; navštíveno 11.02.2018].
URL https://stackoverflow.com/questions/1593976/benefits-of-mvvm-over-mvc

[77] Thomas, E.; Dutta, N.; Lin, C.; aj.: *How to Send Email Using SendGrid with Azure*. Microsoft, Únor 2017, [Online; navštíveno 30.04.2018].
URL https://docs.microsoft.com/en-us/azure/sendgrid-dotnet-how-to-send-email#what-is-the-sendgrid-email-service

[78] Tjaart: *When to favor ASP.NET WebForms over MVC*. StackExchange, Duben 2015, [Online; navštíveno 10.12.2017].
URL https://softwareengineering.stackexchange.com/questions/95212/when-to-favor-asp-net-webforms-over-mvc

[79] Valášek, M. A.: *Stavové HTTP: jak fungují Cookies, Session a ViewState a proč je nepoužívat*. aspnet.cz - in technology we trust, Březen 2008, [Online; navštíveno 19.01.2018].
URL http://www.aspnet.cz/Articles/190-stavove-http-jak-funguji-cookies-session-a-viewstate-a-proc-je-nepouzivat.aspx

[80] Valášek, M. A.: *ViewState: K čemu je a jak ho správně používat*. aspnet.cz - in technology we trust, Červen 2009, [Online; navštíveno 19.01.2018].
URL http://www.aspnet.cz/Articles/235-viewstate-k-cemu-je-a-jak-ho-spravne-pouzivat.aspx

[81] Vice, R.; Siddiqi, M. S.: *MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF*. Birmingham: Packt Publishing, první vydání, 2012, ISBN 978-1-84-968342-5.

[82] Virius, M.: *C# Hotová Řešení*. nám. 28. dubna 48, 635 00 Brno: Computer Press, a.s., první vydání, 2006, ISBN 80-251-1084-2.

[83] Volovec, M.: *Internetové aplikace v .NET Framework*. Bakalářská práce, Vysoké Učení Technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Antonínská 548/1, 601 90 Brno, 2010.

[84] Vrabec, P.: *NA GDPR NEJSOU PŘIPRAVENY ČTYŘI PĚTINY TUZEMSKÝCH ESHOPŮ*. Ekonomický magazín, 2018, [Online; navštíveno 14.04.2018].
URL https://www.ekonomickymagazin.cz/2018/04/na-gdpr-nejsou-pripraveny-ctyri-petiny-tuzemskych-eshopu/

[85] Wagner, B.; Wenzel, M.; Latham, L.; aj.: *What's new in C#*. Microsoft, Leden 2018, [Online; navštíveno 10.02.2018].
URL https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/

[86] Wagner, B.; aj.: *lock Statement (C# Reference)*. Microsoft, Červenec 2015, [Online; navštíveno 30.03.2018].
URL https://docs.microsoft.com/cs-cz/dotnet/csharp/language-reference/keywords/lock-statement

[87] Walls, C.: *Embedded Software: The Works*. Burlington, USA: Elsevier, první vydání, 2006, ISBN 978-0-7506-7954-1.

[88] Walther, S.; Pasic, A.; Dykstra, T.: *Creating Unit Tests for ASP.NET MVC Applications (C#)*. Microsoft, Srpen 2008, [Online; navštíveno 8.01.2018].
URL
https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/unit-testing/creating-unit-tests-for-asp-net-mvc-applications-cs

[89] Wasson, M.: *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET*. Microsoft, Listopad 2013, [Online; navštíveno 10.01.2018].
URL https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[90] Wasson, M.; Latham, L.; Anderson, R.; aj.: *Get Started with ASP.NET Web API 2 (C#)*. Microsoft, Listopad 2017, [Online; navštíveno 18.01.2018].
URL https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api

[91] Wasson, M.; Latham, L.; Pasic, A.; aj.: *Single Page Application: KnockoutJS template*. Microsoft, Leden 2013, [Online; navštíveno 10.01.2018].
URL https://docs.microsoft.com/en-us/aspnet/single-page-application/overview/introduction/knockoutjs-template

[92] Zakas, N. C.; McPeak, J.; Fawcett, J.: *Ajax Profesionálně*. Nové sady 18, 602 00 Brno: ZONER Press, první vydání, 2007, ISBN 978-80-86815-77-0.

[93] Zhang, R.; Kupcik, A.; Milener, G.; aj.: *Transparent data encryption for SQL Database and Data Warehouse*. Microsoft, Říjen 2018, [Online; navštíveno 20.04.2018].
URL https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption-azure-sql?view=azuresqldb-current

[94] informatiky Ministerstvo financí ČR, O.: *Administrativní registr ekonomických subjektů*. Ministerstvo financí ČR, Únor 2013, [Online; navštíveno 05.05.2018].
URL http://wwwinfo.mfcr.cz/ares/

[95] Čápka, D.: *2. díl - Úvod do webových aplikací v ASP.NET*. ITNetwork.cz, Březen 2014, [Online; navštíveno 10.01.2018].
URL https://www.itnetwork.cz/csharp/asp-net/asp-dot-net-tutorial-uvod-do-webovych-aplikaci

[96] Čápka, D.: *Úvod do ASP.NET*. ITnetwork.cz, Leden 2014, [Online; navštíveno 7.12.2017].
URL
https://www.itnetwork.cz/csharp/asp-net/tutorial-uvod-do-asp-dot-net

[97] Čápka, D.: *1. díl - Úvod do Single Page Application v ASP.NET*. ITNetwork.cz, Březen 2016, [Online; navštíveno 10.01.2018].
URL https://www.itnetwork.cz/csharp/asp-net/single-page-application/tutorial-uvod-do-asp-net-single-page-application

Příloha A

Obsah přiloženého paměťového média

```
/mnt/BP_MartinBednar_DVD
├── Konferenční informační systém
│   ├── ConferenceSystem
│   ├── ConferenceSystem.BL
│   ├── ConferenceSystem.DAL
│   ├── ConferenceSystemTests
│   ├── packages
│   ├── ConferenceSystem.sln
│   └── README.md
├── Text bakalářské práce
│   ├── Zdrojový text
│   └── BP_MartinBednar_KonferencniInformacniSytem.pdf
└── Ukázkové aplikace
    ├── ASP.NETMVCApp
    ├── ASP.NETSinglePageApplicationApp
    ├── ASP.NETWebAPIApp
    ├── ASP.NETWebFormsApp
    ├── ASP.NETWebPagesApp
    └── DotVVMApp
```

Adresář *Konferenční informační systém* obsahuje veškeré zdrojové soubory ke stejnojmenné webové aplikaci, která je výsledkem praktické části bakalářské práce. V něm je obsažen důležitý soubor *ConferenceSystem.sln* který zastupuje celé řešení a přes který je možné celou webovou aplikaci otevřít ve vývojovém prostředí Visual Studio. Dále jsou zde obsaženy adresáře jednotlivých projektů: *ConferenceSystem* (prezenční vrstva), *ConferenceSystem.BL* (aplikační logika), *ConferenceSystem.DAL* (datová vrstva), *ConferenceSystemTests* (automatické jednotkové testy). Posledním důležitým souborem je *README.md*, který nese informace o ukázkové verzi webové aplikace.

Adresář *Text bakalářské práce* obsahuje soubor ve formátu *pdf*, který odpovídá vytištěné verzi a také složku *Zdrojový text*, ve které jsou uloženy veškeré zdrojové texty, na základě nichž byl vytvořen soubor *BP_MartinBednar_KonferencniInformacniSystem.pdf*.

Poslední podadresář v kořenovém adresáři DVD nese název *Ukázkové aplikace* a jsou v něm uloženy ukázkové aplikace vytvořené v rámcích a technologiích představených v první

kapitole práce. Název podsložek odpovídá názvu využitého rámce pro danou aplikaci. Spuštění a nasazení těchto ukázkových aplikací není zautomatizované jako nasazení konferenčního informačního systému a může být potřeba manuálně vytvořit, napojit a daty naplnit databázi.